Serveurs vidéo : concepts de base et prototypes

Alice Bonhomme, Ahmed Mostfeaoui

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Serveurs vidéo : concepts de base et prototypes

Alice Bonhomme
LIP/LHPC, ENS Lyon

Ahmed Mosfeaoui
LIP, ENS Lyon

No 3837

December 1999

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--3837--FR

RHÔNE-ALPES

Serveurs vidéo : concepts de base et prototypes

Alice Bonhomme *
LIP/LHPC, ENS Lyon

Ahmed Mosfeaoui †
LIP, ENS Lyon

Thème 1 — Réseaux et systèmes
Projet ReMaP

Rapport de recherche n ° 3837 — December 1999 — 43 pages

Résumé : Ce document présente un état de l'art des techniques utilisées dans la conception de serveurs vidéo ainsi qu'un panorama des différents prototypes développés dans ce domaine. A partir d'un modéle fonctionnel de serveur vidéo, les différents composants d'un serveur vidéo sont décrits : le système de stockage, la gestion mémoire, l'interface avec le client, la gestion des ressources, etc. Les principales techniques implémentant ces composants sont expliquées.

L'ensemble des références aux prototypes et aux articles cités est accessible à l'adresse : `http://www.ens-lyon.fr/~abonhomm/video/survey.html`.

Mots-clé : Serveur vidéo. état de l'art. prototype.

(Abstract: pto)

* Laboratoire pour les Hautes Performances en Calcul (LHPC), École normale supérieure de Lyon, F-69364 Lyon, France. Email: `Alice.Bonhomme@ens-lyon.fr`

† Laboratoire de l'Informatique du Parallélisme, École normale supérieure de Lyon, F-69364 Lyon, France. Email: `Ahmed.Mostefaoui@ens-lyon.fr`

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN (France)
Téléphone : 04 76 61 52 00 - International : +33 4 76 61 52 00
Télécopie : 04 76 61 52 52 - International : +33 4 76 61 52 52

Video servers : design issues and prototypes

Abstract: This report presents a survey of techniques used in the conception of a video server. It also includes a survey of prototypes developped in this area. Based on a fonctionnal model of video server, the different components of a video server are described: the storage system, the memory management, the client interface, the ressources management, etc. The main techniques that implement these components are explained.

The prototypes and articles references can be found at the following address :
`http://www.ens-lyon.fr/~abonhomm/video/survey.html`.

Key-words: Video server. survey. prototype.

Table des matières

Introduction

L'avancée technologique, enregistrée depuis quelques années dans différents domaines de l'informatique (puissance de calcul des processeurs, capacité de stockage des disques, bande passante des réseaux, etc.), a permis l'émergence d'une nouvelle classe d'applications appelées **applications multimédia**. L'avènement du multimédia, conjugué à la vulgarisation rapide de l'Internet, reste certainement l'un des événement les plus marquants durant la dernière décennie dans le domaine de l'informatique, en particulier pour le grand public. Un grand nombre d'applications multimédia, en particulier les applications vidéo, ont ainsi vu le jour aussi bien sous la forme d'applications professionnelles telles que la vidéo à la demande (*Video-On-Demand, VOD*), la gestion d'archives audiovisuelles, etc. que sous la forme d'applications grand public telles que les boutiques virtuelles, etc.

Parmi ces applications multimédia, celles qui suscitent le plus d'intérêt sont certainement les applications vidéo. Ceci est motivé par deux facteurs principaux. Tout d'abord, le marché de la vidéo engendre des enjeux économiques non négligeables. Nombre d'industriels et de professionnels de l'informatique ont en effet substantiellement investi dans les applications vidéo [Nat95] grand public comme la VOD. D'autre part, la conception de systèmes vidéo, indépendamment de leurs domaines d'application, impose de nouveaux défis en termes d'efficacité, de robustesse et surtout de faisabilité économique.

Au cœur de tout système vidéo, le *serveur vidéo* constitue la partie centrale responsable du stockage et de la gestion des données audiovisuelles. Les données audiovisuelles ont une nature intrinsèquement différente de celles des données dites conventionnelles (texte, image, etc.), leur gestion impose souvent le développement de techniques spécifiques.

L'objectif de ce document est de présenter les différents concepts de base d'un serveur vidéo commun à la majorité des applications vidéo. Les techniques utilisées sont étroitement dépendantes des applications visées. Les approches utilisées dans l'ordonnancement des bras de lecture des disques par exemple influencent la manière avec laquelle la mémoire est gérée qui, à son tour, influence les techniques de communication avec le client. Ainsi, par souci de clarté, nous commençons d'abord par adopter une architecture fonctionnelle type composée de plusieurs parties communes à la majorité des serveurs vidéo (section 2). Cette architecture logicielle permettra, par la suite, l'introduction des différentes techniques utilisées au niveau de chaque partie du serveur.

Nous suivons le cheminement d'une requête de lecture d'un flux depuis sa réception par le serveur jusqu'à l'envoi du flux demandé au client [1] et nous décrivons les différents mécanismes impliqués. Dans la section 3, nous étudierons le gestionnaire des ressources, responsable du fonctionnement du serveur. La section 4 présente le système de stockage d'où les flux vidéo sont envoyés. La section 5, étudie les techniques de gestion de la mémoire pour les flux continus. Dans la section 6, les paradigmes de communication avec le client sont exposés. Enfin, la section 7 présente les choix architecturaux des plates-formes matérielles retenues pour servir de base au développement des serveurs vidéo. La section 8 fait un état de l'art des prototypes développés depuis quelques années. Pour conclure la section 9 discute des perspectives pour les serveurs vidéo.

1 Contexte d'étude

1.1 Caractéristiques des données audiovisuelles

Volume important

A l'inverse des données textuelles, les données audiovisuelles digitalisées présentent des volumes très importants [GVK+95]. A titre d'exemple, une seconde de vidéo haute définition (TVHD) (voir table 1.1) nécessite un volume de 351 Mo ! Des techniques de compression de données vidéo [2] ont donc été développées afin d'en réduire le volume (format de compression MPEG [Gal91] par exemple). Néanmoins, même après compression, les données vidéo demeurent volumineuses. Face à un tel volume, les techniques de stockage

1. Dans le reste du document, le terme client est utilisé pour désigner l'application réceptrice du flux, que ce soit un simple terminal de visualisation ou une application complexe.

2. Dans le reste du document, les termes "données vidéo" et "données continues" sont interchangeables et font référence à des données audiovisuelles.

classiques s'avèrent inefficaces. Il convient donc de développer, d'une part, des systèmes de stockage adaptés (souvent hiérarchisés) et, d'autre part, des politiques de gestion efficaces.

Spécifications	volume nécessaire
Qualité TVHD (1024 × 2000 pixels/frame, 24 bits/pixel)	351 Mo/sec
Qualité NTSC (640 × 480 pixels/frame, 24 bits/pixel)	27 Mo/sec
MPEG2	4 Mbits/sec
MPEG1	1.5 Mbits/sec
Qualité CD audio	1.4 Mbits/sec

TAB. 1 – *Volume requis par les données continues.*

Données isochrones

Une autre caractéristique contraignante des données continues est leur *isochronisme*. Une séquence vidéo ne peut en effet être compréhensible que si les images dont elle est composée, sont affichées à une vitesse constante. Par exemple, pour le standard européen (PAL/SECAM), la vitesse est de 25 images/seconde alors qu'elle est de 30 images par seconde pour le standard américain (NTSC). Cette dimension temporelle, appelée aussi contrainte temps réel ou *isochronisme*, impose de facto des contraintes de délai qu'il faut respecter à tous les niveaux du cheminement de l'information vidéo, depuis sa source jusqu'à son point de consommation.

Présentation synchronisée

De manière générale, une information vidéo est souvent associée à plusieurs informations d'autres média (audio, texte, image, etc.). Lors de la présentation de l'ensemble de ces informations, le système doit être capable de les synchroniser afin de constituer une information cohérente. Par exemple, il n'est pas aisé de comprendre un film dont la bande son ou le sous-titrage sont décalés, même de quelques secondes, par rapport à la vidéo. Il est donc impératif que le système soit capable d'assurer la synchronisation entre plusieurs média (vidéo, images, texte, etc.) tout en tenant compte de leurs caractéristiques spécifiques respectives.

Compression irrégulière

Comme mentionné précédemment, les techniques de compression permettent de réduire considérablement le volume des données vidéo. Ces techniques s'appuient sur le principe de la ressemblance entre les images successives d'une vidéo. Nous avons deux types de compression: les compressions CBR (*Constant Bit Rate*) et les compressions VBR (*Variable Bit Rate*). Dans le cas d'une compression CBR, les différences entres les images successives sont codées par des *frames* de taille constante. Pour une compression VBR, les *frames* peuvent être de taille variable en fonction des séquences vidéo plus ou moins riches en mouvement. Cependant, il est très difficile d'appréhender la variabilité des VBR d'autant que le système doit en assurer l'isochronisme. Pour cette raison, une majorité des travaux de recherche font l'hypothèse que les données après compression demeurent avec un débit constant, CBR. Dans ce document, on s'intéresse essentiellement aux CBR, sauf mention explicite pour les VBR, pour simplifier l'introduction des différents concepts.

1.2 Applications multimédia

La conception et la mise en œuvred'un serveur vidéo sont intrinsèquement liées aux caractéristiques de l'application multimédia cible. Les applications multimédia ont en effet des exigences et des besoins très différents qui influencent, de manière directe, les techniques de gestion interne aux serveurs vidéo. Afin de simplifier l'introduction des différents concepts de base liés aux serveurs vidéo, commençons d'abord par présenter un panorama de classes d'applications multimédia en mettant en relief leurs caractéristiques.

Applications *Vidéo-à-la-demande* (VOD[3]) : les données accédées dans cette classe d'applications sont de longue durée (film de cinéma ou cours de chirurgie par exemple) et sont relativement indépendantes les unes des autres. Elles ne possèdent pas ou peu de liens structurels et sémantiques qui les relient (films d'une vidéothèque par exemple). Cependant, le scénario d'accès, dans ce genre d'applications, est peu interactif (utilisation peu fréquente des fonction de magnétoscope[4]).

Applications *News-On-Demand* (NOD) : dans cette classe d'applications, les données accédées ont des longueurs relativement courtes (de quelques secondes à quelques minutes). Généralement, les données accédées sont fortement structurées (par exemple, par les liens structurels et sémantiques entre les informations politiques). Les scénarios d'utilisation sont fortement interactifs via des mécanisme de recherche ou l'utilisation de fonctions magnétoscope. Cette classe d'applications regroupe, outre les applications NOD, les applications de boutique virtuelle, de gestion d'archives audiovisuelles, etc.

Applications interactives: cette classe regroupe essentiellement les applications de jeux vidéo ou de réalité virtuelle. Contrairement aux classes d'applications précédentes dans lesquelles les données audiovisuelles accédées possèdent un débit connu à l'avance (défini par le film ou le clip vidéo), dans cette classe d'applications, la dimension temporelle des données dynamiques est inconnue à priori i.e., elle est régie par l'interaction avec le client. L'animation des objets, dans un jeu vidéo par exemple, est tributaire à la fois des choix du joueur et de son état d'avancement dans le jeu.

1.3 Approche de séparation ou approche d'intégration?

On peut distinguer, dans une application multimédia, deux classes de données: les données statiques (texte, images, ...) et les données dynamiques (audio, vidéo, animation, ...). Ces deux types de données, en raison de leur nature intrinsèquement différente, nécessitent des techniques de gestion spécifiquement adaptées. Deux approches, dans la conception des systèmes multimédias, sont ainsi envisageables (figure 1).

Approche de séparation

Dans cette approche, le système multimédia est constitué de deux sous-systèmes distincts. Un premier sous-système est dédié à la gestion des données statiques. Toutes les techniques issues du domaine de la gestion des bases de données classiques peuvent être utilisées. Un deuxième sous-système est, quant à lui, dédié à la gestion des données dynamiques (c-à-d. ayant des contraintes temps réel). Ce système est appelé communément le serveur vidéo. Pour procurer une vue transparente à l'application, une couche logicielle d'intégration assure la coordination entre les deux sous-systèmes. Cette approche offre l'avantage d'une implantation relativement moins complexe, d'une meilleure évolutivité du système et de meilleures performances puisque les techniques de gestion peuvent être adaptées à chaque classe de données.

Approche d'intégration

A l'inverse de l'approche précédente, les deux classes de données, dans cette approche, coexistent dans un seul système. Ce dernier doit assurer de façon interne une gestion appropriée à chaque classe de données. Cette approche offre l'avantage d'un bon équilibrage de charge dans le système et d'une utilisation rationnée des ressources de ce dernier. En effet, lors d'une forte charge d'accès sur une classe de données (données audiovisuelles par exemple), le système peut mobiliser la totalité de ses ressources alors que cela n'est pas possible dans l'approche précédente. Cette approche est surtout adaptée aux applications très interactives comme la réalité virtuelle dans lesquelles les données accédées ainsi que le scénario d'interaction ne sont pas connus à l'avance (voir le serveur *Symphony*, section 8.3.3).

3. *Video-On-Demand.*
4. Avance rapide, recul rapide, pause, retour au début, avance à la fin, etc.

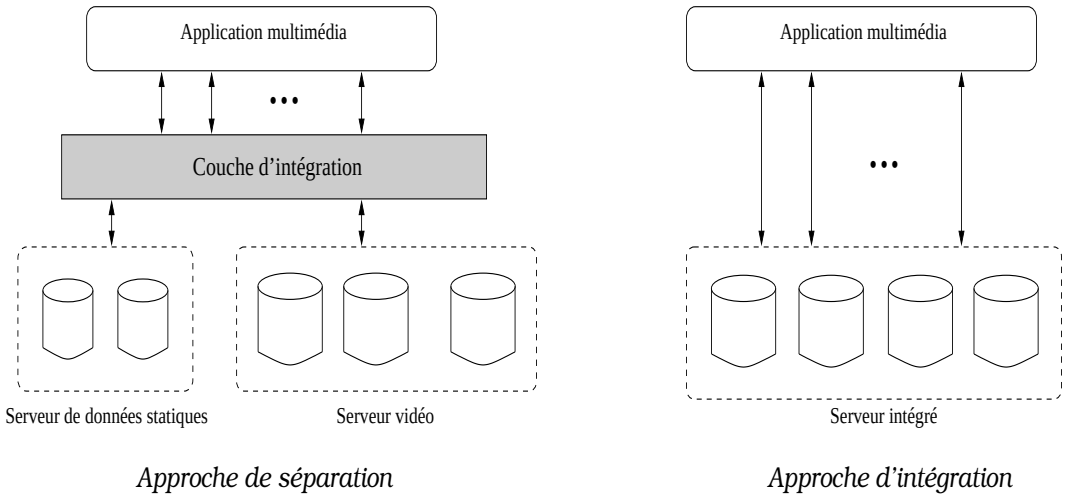

FIG. 1 – *Deux approches de conception de systèmes multimédias: approche de séparation et approche d'intégration.*

1.4 Modes d'interaction : *server push* ou *client pull* ?

Il existe deux paradigmes d'interaction dans un environnent client/serveur pour les applications vidéo [SGV95, Lee98] , dits *"client pull"* et *"server push"*, illustrés dans la figure 2.

Client pull

Dans ce paradigme, la gestion de l'isochronisme des données vidéo est laissée à la charge du client. Ainsi, l'accès à un flux vidéo se traduit par un ensemble de sous-requêtes que le client envoie au serveur à chaque fois qu'il a besoin de blocs de données (voir figure 2) d'où l'appellation *"client pull"*. Le serveur est donc déchargé de la contrainte temps réel et est réduit à assurer des temps de réponse minimaux.

Ce paradigme présente l'avantage d'une meilleure gestion des ressources client et serveur: le client ne demande que ce dont il a besoin, c'est-à-dire qu'il réduit par exemple l'espace mémoire requis puisqu'il intègre directement la variabilité des flux (VBR). La gestion des fonctions magnétoscope (pause, stop, avance et retour rapide, etc.), qui souvent se traduisent par des accès à des blocs de données non consécutifs dans le flux, est aussi grandement facilitée.

Cependant, toutes les techniques d'optimisation présentées dans les sections suivantes, qui tirent profit de la périodicité des données vidéo, ne trouvent pas leur justification dans ce paradigme, ce qui a des incidences directes sur les performances du serveur. De plus, ce paradigme, du fait du dialogue permanent entre le client et le serveur, génère un trafic de communication plus important dans le sens client vers serveur. Pour ces raisons, ce paradigme reste peu utilisé par les applications vidéo.

Server push

A l'inverse de l'approche précédente, la gestion de l'isochronisme des flux vidéo est à la charge du serveur. Un client envoie donc une seule requête d'accès à un flux et c'est au serveur de gérer et de transmettre le flux correspondant en s'assurant de sa continuité (voir figure 2). Les nombreuses techniques d'optimisation décrites dans les sections suivantes peuvent être appliquées, améliorant ainsi les performances du serveur. Ce type de fonctionnement a été largement adopté dans la littérature et la quasi-totalité des prototypes de serveurs vidéo développés sont basés sur ce principe.

Néanmoins, la gestion de l'isochronisme des flux au niveau du serveur constitue une tâche complexe du fait que ce dernier doit maintenir en permanence l'état des clients. En outre, le support des fonctions magnétoscope est difficile à mettre en œuvre en raison de la variabilité des données vidéo compressées.

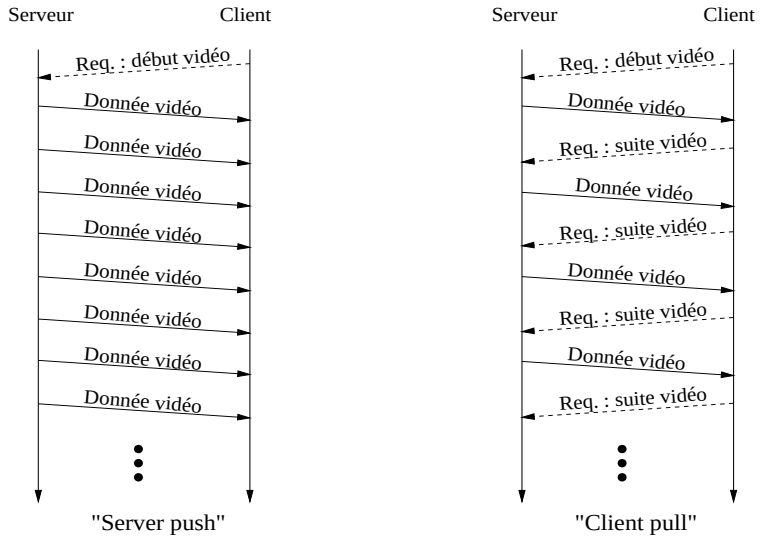

FIG. 2 – *Deux paradigmes d'interaction entre serveur et client.*

En résumé, le paradigme *"server push"*, comparé au paradigme *"client pull"*, permet de meilleures performances au détriment d'une gestion plus complexe. Dans ce document, nous nous intéressons au mode *server push*.

2 Modèle fonctionnel de serveur vidéo

L'objectif de ce document est de présenter de manière synthétique les concepts liés aux serveurs vidéo sans se focaliser particulièrement sur une architecture matérielle spécifique ou des techniques adaptées à une classe d'applications vidéo. Nous introduisons donc un modèle fonctionnel d'architecture de serveur vidéo(figure 3). Ce modèle reste évidemment assez général pour être appliqué à tout type de plate-forme. On distingue dans ce modèle quatre composants: (a) le gestionnaire de ressources, (b) le gestionnaire de stockage, (c) le gestionnaire de mémoire et (d) le gestionnaire d'interface. Ces composants ne sont pas totalement indépendants les uns des autres. Pour cette raison, il n'existe pas de techniques optimales propres à chaque composant, mais plutôt un compromis global.

Le gestionnaire de ressources est le composant responsable de la gestion interne du serveur. Il contrôle tous les flux de données des clients acceptés. Il exerce également un contrôle d'admission vis-à-vis des nouveaux clients afin de s'assurer que les ressources du serveur suffisent à supporter l'ensemble des flux simultanés. A cet effet, il échange des données de contrôle avec les trois autres modules (voir figure 3).

Le gestionnaire de mémoire est responsable du stockage temporaire des données audiovisuelles en provenance du gestionnaire de stockage et destinées au gestionnaire d'interface. Il est notamment responsable des techniques de partage de flux.

Le gestionnaire de stockage assure le stockage et le chargement transparent des données vidéo. De manière à paralléliser les utilisations des ressources matérielles, on utilise un mécanisme de *pipeline* supervisé par le gestionnaire de ressources et dont les étages sont formés par les trois autres modules.

Le gestionnaire d'interface est le composant responsable de l'interaction avec le client. Il reçoit les requêtes du client et envoie les données aux clients via le réseau de distribution.

Les techniques de gestion relatives à ces quatre composants sont présentées dans le reste de ce document. Notons qu'en fonction des ressources dont il dispose, un serveur vidéo peut gérer plusieurs dizaines voire plusieurs centaines de requêtes simultanément. Pour ce faire, son fonctionnement interne est basé sur une

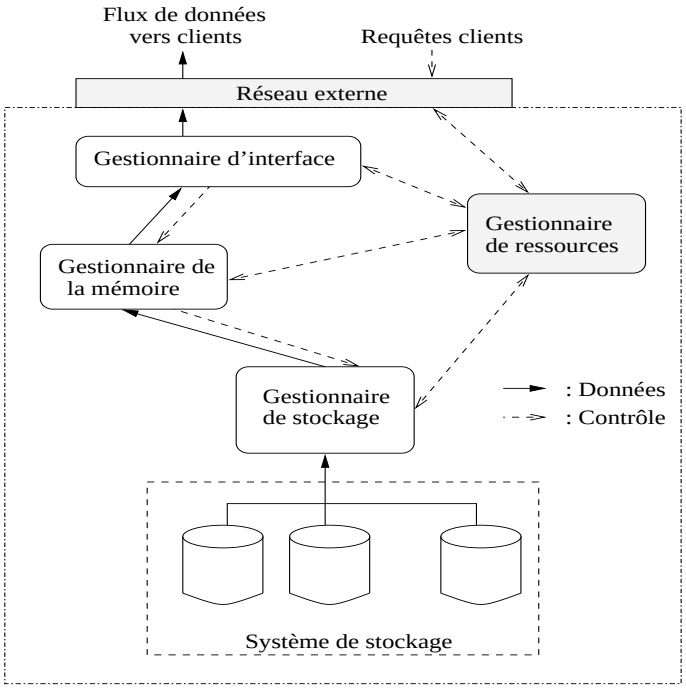

FIG. 3 – *Un modèle fonctionnel de serveur vidéo.*

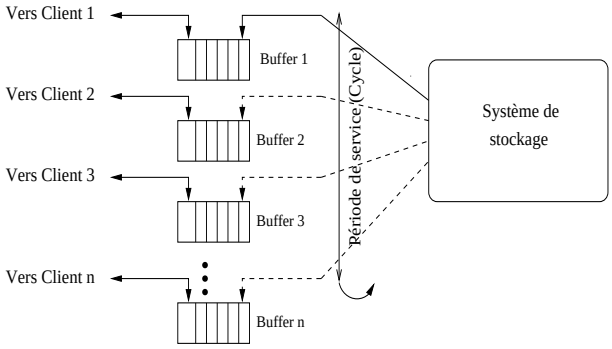

FIG. 4 – *Fonctionnement en cycle d'un serveur vidéo. Durant un cycle, le serveur charge, pour chaque flux, des données depuis le système de stockage vers la mémoire allouée à cet effet. Les données chargées sont ensuite consommées par les clients*

allocation périodique des ressources aux différentes requêtes acceptées dans le serveur de façon à assurer à chacune d'elles une continuité de service. La figure 4 schématise ce fonctionnement. La période de service est appelée **cycle** du serveur. Durant un cycle, le serveur charge, pour chaque flux, des données depuis le système de stockage vers la mémoire allouée à ce flux de façon à ce que la quantité de données consommée par ce flux pendant un cycle n'excède pas la quantité de données chargées.

La taille de la mémoire réservée pour chaque requête ainsi que la bande passante du système de stockage conditionne la longueur de cycle d'un serveur. Ces trois paramètres (cycle du serveur, mémoire disponible et bande passante du système de stockage) déterminent la capacité d'un serveur vidéo évaluée en terme de nombre de requêtes supportées simultanément. Ces paramètres seront étudiés dans les sections suivantes.

3 Gestion des ressources

L'objectif premier d'un serveur vidéo est de servir un maximum de clients simultanément, tout en leur garantissant une bonne qualité de service. Cette dernière se traduit, à ce niveau, par la faculté du serveur vidéo à assurer des flux continus, sans interruption, à tous les clients admis dans le système. Néanmoins, les ressources du serveur (espace mémoire, bande passante des disques, etc.) sont limitées et partagées entre les différents flux des clients. Il incombe donc au gestionnaire de ressources d'en contrôler le nombre et l'utilisation afin de garantir à chacun le service auquel il a souscrit.

Dans les applications vidéo, les accès aux données sont souvent biaisées, c'est-à-dire qu'il est fréquent qu'une grande partie des clients accède à un ensemble restreint de données. Le rôle du gestionnaire de ressources est aussi d'optimiser ces accès en essayant de faire partager les données entre les différents accès. Cette tâche d'optimisation est essentielle dans les serveurs vidéo compte tenu du volume que représentent les données vidéo. Les tâches du gestionnaire de ressources se résument donc, en premier lieu, à contrôler l'accès des clients aux données dans le serveur vidéo et, en second lieu, à optimiser les accès de plusieurs clients au même flux. Dans le reste de cette section, on présente d'abord un panorama des techniques utilisées par le gestionnaire de ressources en matière de contrôle d'admission suivi des techniques d'optimisation, principalement l'ordonnancement des requêtes. Finalement, une discussion sur la qualité de service à ce niveau clôt cette section.

3.1 Contrôle d'admission

La notion de contrôle d'admission dans un serveur vidéo est liée à l'affectation des ressources du serveur aux différents clients. Ce mécanisme permet de prendre la décision d'accepter ou non un nouveau client. A l'inverse des serveurs de fichiers classiques, les données vidéo, par leur isochronisme, imposent des délais stricts d'acheminement des données aux clients que le serveur vidéo doit respecter. Ces délais se répercutent jusqu'au système de stockage. L'objectif du contrôle d'admission est donc de s'assurer que ces délais soient respectés pour l'ensemble des clients admis dans le serveur. On distingue, dans la littérature, trois catégories de contrôle d'admission: (a) les contrôles déterministes qui garantissent des flux continus sans interruption à la sortie du serveur aux clients admis, (b) les contrôle statistiques qui assurent un service avec une certaine probabilité de réussite et (c) les contrôles dits *"meilleur effort"* qui ne donnent aucune garantie sur le service auquel le client a souscrit.

Contrôle d'admission déterministe

Les premiers travaux, dans la littérature, sur le contrôle d'admission dans les serveurs vidéo se sont concentrés, pour la plupart, sur les contrôles déterministes [GIzS98, CC96]. La plus grande partie des prototypes implantés utilise ce type de contrôle d'admission. Un contrôle d'admission déterministe assure un flux continu sans perturbation pour chaque client présent dans le serveur. Ainsi, lors de la soumission d'une nouvelle requête, le contrôleur vérifie que le futur flux généré par l'acceptation de cette requête n'altérera pas les flux déjà acceptés. En d'autres termes, le gestionnaire de ressources s'assure qu'il y a assez de ressources disponibles dans le serveur pour servir l'ensemble des requêtes. Or, les ressources du serveur varient en fonction de l'architecture matérielle sous-jacente (bande passante disque, CPU, bande passante réseau si on a un stockage distribué, etc.) qui détermine les délais d'acheminement des blocs de données depuis l'espace

de stockage jusqu'à la mémoire ainsi que les capacités de stockage mémoire différentes. Tous ces paramètres doivent être pris en compte par le gestionnaire de ressources lors de l'admission d'une nouvelle requête. Dans ce qui suit on présente les fondements du contrôle d'admission déterministe.

Supposons qu'à un instant t, le serveur vidéo supporte n flux vidéo, il assure donc un débit constant pour n requêtes $R_0, \ldots, R_{n-1}$ de débit constant $r_0, \ldots, r_{n-1}$. Soit R_n une nouvelle requête de débit r_n. Le serveur doit décider si oui ou non la requête peut être acceptée. Notons T la période de service du serveur, en d'autres termes, le serveur sert les requêtes de façon cyclique chaque T unités de temps. Le flux associé à la requête R_i consomme donc $(T * r_i)$ bits par cycle. Ceci implique qu'avant chaque cycle, $(T * r_i)$ bits doivent être rapatriés par le serveur depuis l'espace de stockage vers la mémoire réservée pour ce flux. Supposons que les blocs de données sont de taille b et que la taille mémoire totale disponible dans le serveur en nombre de blocs est de M_{total} blocs, on peut donc énoncer la première condition pour l'acceptation de la requête R_n comme suit:

$$\sum_{i=0}^{n} \left(\lceil \frac{T * r_i}{b} \rceil + 1 \right) \leq M_{total} \tag{1}$$

Cette condition traduit la contrainte de taille mémoire disponible dans le serveur. En effet, l'ensemble de la mémoire nécessaire à toutes les requêtes, pendant un cycle T, doit être inférieure ou égale à la taille mémoire disponible. Le 1 dans la formule est nécessaire au cas où les $(T * r_i)$ bits se trouvent à cheval entre deux blocs mémoires, nécessitant ainsi le chargement des deux blocs. Notons que la taille mémoire nécessaire pour chaque flux est fonction du débit requis par ce dernier et de la période T du cycle du serveur. Si le débit des données vidéo est une caractéristique de ces dernières, et donc indépendant de la conception du serveur, la période de cycle, en revanche, est fixée de manière à assurer un bon équilibre entre le nombre total de flux supportés simultanément et la taille mémoire totale disponible dans le serveur. Plus la période T est longue, plus la taille de la mémoire nécessaire augmente.

Notons $T_{acheminement}$ le temps d'acheminement d'un bloc de données du système de stockage vers la mémoire, y compris le temps de lecture. Ce temps est fonction des temps de rotation des disques, des temps de déplacement de leurs bras de lecture, des temps de transport sur le réseau interne s'il s'agit d'un espace distribué, etc. On peut donc déduire une deuxième condition nécessaire pour l'acceptation de la requête R_n comme suit:

$$\sum_{i=0}^{n} \left(\left(\lceil \frac{T * r_i}{b} \rceil + 1 \right) * T_{acheminement} \right) \leq T \tag{2}$$

Cette condition permet au serveur de s'assurer de la non violation de la contrainte imposée par la bande passante du système de stockage. Le temps total nécessaire à l'acheminement des blocs de données de toutes les requêtes acceptées doit en effet rester inférieur ou égal à la période de service T.

Globalement, il s'agit ici de déterminer de façon optimale la valeur de T pour un maximum de flux simultanés tout en prenant en compte les contraintes imposées par la mémoire disponible (condition 1), d'une part, et par la bande passante du système de stockage (condition 2), d'autre part.

Lors de la soumission d'une nouvelle requête au serveur, le gestionnaire de ressources vérifie si les deux conditions, 1 et 2, ne sont pas violées. Si c'est le cas, la requête est acceptée et intégrée dans le serveur, sinon la requête est rejetée.

Contrôle d'admission statistique

La banque de données de vidéos stockées dans un serveur vidéo contient en général des vidéos de format divers et par conséquent de débits différents, ainsi que souvent des vidéos VBR. De plus, nombre d'applications vidéo ont une certaine tolérance à la perte de paquets de données. Une application comme la téléconférence, par exemple, accepte que quelques images soient altérées sans pour autant que cela dégrade la qualité de la transmission dans son ensemble. En d'autres termes, on peut tolérer, au niveau du serveur, que certains blocs ne sont pas envoyés à temps.

L'idée à la base des contrôleurs d'admission statistiques, consiste, à l'instar des contrôleurs déterministes, de tirer profit de ces deux aspects de façon à ne pas considérer les valeurs maximums des débits, mais à

trouver une distribution statistique des débits permettant d'assurer une qualité de service avec une probabilité de réussite proche de l'optimal [VPGG94, CZ96]. Ainsi le serveur vidéo garantit, pour chaque client, qu'il recevra, dans les délais requis, un pourcentage minimum (95% par exemple) de l'ensemble des blocs de données demandés.

Cependant, dans ce type de contrôle d'admission, on se trouve face à un risque d'avarie. En effet, si les blocs de données, n'ayant pu être acheminés dans les délais, se suivent (c'est-à-dire sont consécutifs dans la vidéo), la dégradation de la qualité de service peut devenir inacceptable et ce, même si le pourcentage de réussite est respecté. Pour pallier ce problème, Vin et al. [VPGG94] proposent de distribuer les risques de non acheminement des blocs de données sur l'ensemble des flux. Ainsi, les risques d'accumulation des blocs non acheminés sont diminués et la qualité de service est maintenue.

Vis-à-vis de certaines applications, un contrôle d'admission statistique s'avére plus pertinent que le contrôle déterministe car il permet de servir plus de clients simultanément tout en leur offrant un service satisfaisant.

Contrôle d'admission "meilleur effort"

La troisième classe des contrôleurs d'admission ne donne, *a priori*, aucune garantie sur le service [VGGG94]. Les requêtes sont acceptées et servies uniquement si le système a des ressources disponibles à un instant donné. Ce type de contrôle d'admission est utilisé en complément des deux premiers. En effet, il arrive qu'à cause d'un changement de débit des données vidéo, le serveur dispose de ressources disponibles pendant un certain laps de temps. Celles-ci peuvent alors être allouées aux clients qui ont souscrit à ce type de service.

3.2 Ordonnancement des requêtes

Dans les applications vidéo, les accès vers le serveur vidéo ne sont pas uniformément distribués sur l'ensemble des données vidéo disponibles. Il en découle qu'un certain sous-ensemble de données est généralement plus fréquemment accédé que le reste des données, par exemple dans les applications de VOD ou de téléachat. Les films récents ou les articles à la mode (durant la fête des mères par exemple) sont particulièrement demandés par rapport au reste des données. Partant de ce constat, il paraît pertinent de faire partager des blocs en mémoire entre plusieurs clients accédant au même flux. Le paradigme de partage de données entre clients constitue un point d'intérêt particulier pour les applications vidéo vu le volume des données manipulées et les ressources requises. Le partage peut être total (un flux est alors partagé entre plusieurs clients) ou partiel (seuls certains fragments de données sont partagés). Par la suite, on s'intéresse plus particulièrement au partage total. Le partage partiel sera étudié dans la section 5.1.

Les premiers travaux, dans la littérature, relatif au partage de données dans les serveurs vidéo ont principalement concerné le partage d'un même flux entre différents clients. Cette technique est connue sous le nom de regroupement des accès (*"Batching"*) [SY98, JSC97, DSS96]. L'intérêt du regroupement des accès est de servir plusieurs clients par le même flux vidéo sans introduire de charge supplémentaire au niveau du serveur. Des mécanismes de diffusion, intégrés dans les nouveaux protocoles réseaux (ATM par exemple [Bou92]), permettent en effet d'assurer l'acheminement d'un même flux en sortie du serveur vidéo vers plusieurs clients. La capacité du serveur est ainsi augmentée en terme de nombre de clients supportés simultanément. Cependant, son efficacité n'est effective que sous les deux conditions suivantes.

Distribution biaisée des accès. Il existe, parmi les données disponibles dans le serveur, des données *chaudes* (fréquemment accédées) et des données *froides* (rarement accédées).

Tolérance à l'attente. Afin d'assurer un regroupement optimal des accès, les requêtes doivent pouvoir tolérer un temps d'attente initial permettant au serveur de les regrouper. Cette condition varie considérablement en fonction de l'application cible. Si dans la VOD, les clients peuvent tolérer des temps d'attente de l'ordre de quelques minutes, dans une application de gestion d'archives en revanche, les temps d'attente tolérés sont très courts (de l'ordre de quelques secondes au plus).

La difficulté majeure dans la conception d'un ordonnanceur utilisant une technique de regroupement réside donc dans une bonne appréhension de ces deux conditions. En effet, d'une application à une autre, la distribution des accès et la tolérance d'attente des requêtes varient considérablement. Cette variation est

également perceptible à l'intérieur d'une même application. Par exemple, dans une application VOD, il est fréquent que la distribution et la fréquence d'accès aux films change rapidement pendant les soirées [LV94]. Si un ordonnanceur alloue les ressources aux premiers clients sans tenir compte des clients qui suivent, il risque de saturer rapidement le serveur sans pouvoir profiter de l'opportunité du regroupement. A l'inverse, s'il fait attendre les clients trop longtemps au vu d'un éventuel regroupement, certains clients risquent d'abandonner leur requête.

La gestion de ce regroupement est rendue plus complexe par d'éventuelles interactions avec les clients. En effet, si l'application supporte les fonctions de magnétoscope (avance rapide, recul rapide, pause, etc.), l'intervention d'un client sur un flux risque de perturber la synchronisation avec les autres clients partageant ce même flux. Cela nécessite des ressources supplémentaires au serveur pour gérer cette désynchronisation [DS96].

Compte tenu du comportement "chaotique" des clients, les techniques d'ordonnancement de requêtes exploitant le regroupement, ont été en majorité fondées sur des politiques *run-time*.

Politique FCFS (*First Come First Served*). Dans cette politique, toutes les requêtes en attente sont mises, selon leur ordre d'arrivée, dans la même file, appelée *file d'attente*. Dès la libération de ressources suffisantes, l'ordonnanceur choisit immédiatement la requête en tête de file pour la servir. Toutes les requêtes de la file, accédant au même flux que la requête de tête, sont sélectionnées pour le regroupement. A première vue, cette politique n'exploite pas directement la distribution des accès aux données puisqu'aucune priorité n'est assignée aux requêtes accédant les données chaudes. En contrepartie, elle assure que toutes les requêtes seront servies.

Politique MQL (*Maximum Queue Length*). Cette politique affecte à chaque vidéo disponible dans le serveur une file d'attente dans laquelle les requêtes l'accédant sont mises en attente. La sélection des requêtes à servir, après libération de ressources, est basée sur la cardinalité des files d'attente, en d'autres termes, la file contenant le maximum de requêtes en attente est servie en priorité. Cette politique est similaire à la politique **MRF (*"Maximum Request First"*)** [WA85]. Cette dernière favorise le partage des données en sélectionnant la vidéo présentant un maximum de partage. Néanmoins, dans une configuration de serveur à capacité limitée, elle présente un inconvénient majeur qui est celui du risque de famine. En effet, les données froides risquent de ne pas être servies ou d'être servies après des temps d'attente importants car les files d'attente leur correspondant ont une faible cardinalité.

Politique FCFS-n. Comme on l'a vu, la politique FCFS ne favorise pas explicitement les accès chauds. Pour pallier cet inconvénient, la technique FCFS-n réserve à l'avance les ressources nécessaires pour servir périodiquement les n vidéo les plus demandées [DS96]. Ainsi, les flux vidéo les plus chauds sont envoyés périodiquement (toutes les B minutes par exemple). La période B est appelée *période de regroupement* dans laquelle les requêtes pour la même vidéo sont regroupées. Le serveur garantit ainsi un temps d'attente maximal pour les requêtes accédant les vidéo chaudes. Il faut noter qu'avec la valeur de n égale à 0 (FCFS-0), la politique FCFS-n est réduite à une politique FCFS simple. Les requêtes accédant les données froides, quant à elles, sont servies en FCFS classique. Ainsi, les ressources du serveur sont divisées en trois catégories: les ressources réservées pour les n vidéos chaudes, les ressources *allouées à la demande* pour les vidéo froides et les ressources de *contingence* dédiées aux opérations de désynchronisation dues à une interaction dans un flux partagé [DS96]. L'efficacité de cette politique réside alors dans le bon choix du paramètre n et du partitionnement judicieux des ressources.

D'autres politiques d'ordonnancement, variantes des techniques exposées ci-dessus, ont été proposées dans la littérature, par exemple, la technique dite *"wait tolerance batching schemes"* [YWS96] ou la technique LASS (*Look-Ahead Stream Scheduling*) [YWS96]. Cette dernière intègre des mécanismes pour la prise en compte des problèmes de désynchronisation dans les flux partagés dus à l'intervention des clients.

Ces techniques d'ordonnancement ont été en majorité développées pour les applications VOD. Il s'agit dans ces applications de faire partager des flux vidéo d'une longueur importante (se comptant en heures) entre des clients peu utilisateurs des boutons magnétoscope (l'intervention des clients est plutôt rare). Néanmoins, ces hypothèses ne sont pas valides dans d'autres applications comme les applications *news-on-demand* ou la gestion d'archives vidéo. En effet, en raison de la taille des données vidéo manipulées dans ces applications

(la longueur d'un clip vidéo reste en dessous de 5 minutes en général), les scénarios d'accès aux requêtes varient également. Généralement, ce genre d'applications est sujet à des pics d'accès.

Il apparaît alors que d'autres techniques de partage de données entre clients sont nécessaires: le partage ne concerne plus des flux entiers, mais il se focalise sur des portions de flux. Ce partage, appelé partage partiel, est géré en général au niveau de la mémoire. Nous étudierons ces techniques dans la section 5.1 sur la gestion de la mémoire.

3.3 Qualité de service

La notion de qualité de service (*Quality Of Service*, QOS) fait souvent référence aux propriétés du réseau de communication reliant les clients avec le serveur. Dans ce cadre, une grande partie des travaux de recherche s'est concentrée sur le développement de protocoles de communication adaptés au transport de flux vidéo [CLY96]. L'objectif de ces protocoles est d'assurer d'une part une bande passante suffisante pour le transport isochrone des données vidéo et d'autre part, une qualité de transport satisfaisante en minimisant les risques de perte de paquets de données.

En réalité, la qualité de service ne concerne pas uniquement le domaine de la communication, mais elle est souvent intrinsèque à l'application elle-même [CCH94]. Nous illustrons nos propos par l'exemple suivant. On considère une application d'enseignement assistée par ordinateur dans laquelle un ensemble de cours, dans différentes disciplines, est disponible sous forme de vidéo en accès distant à l'ensemble des étudiants d'une université. Il apparaît clair que les besoins des étudiants au niveau de la qualité de la vidéo (qualité de compression) et de l'interaction avec le système ainsi que leur tolérance aux mauvaises transmissions varient en fonction de leur discipline. Des étudiants en médecine, visualisant une opération à coeur ouvert, sont en effet plus exigeants et moins tolérants que des étudiants suivant un cours magistral ! Il est donc évident que la gestion de la qualité de service doit être prise en compte au niveau du serveur de l'application elle-même. Cette tâche peut être aussi bien rattachée au gestionnaire d'interface qu'au gestionnaire de ressources. Il faut noter que la qualité de service requise par différents clients d'une application peut avoir des incidences sur les ressources du serveur et par conséquent sur les différentes politiques de gestion interne (ordonnancement des accès, gestion de la mémoire, etc.) [CLY96, CCH94]. Lors de la conception d'un serveur vidéo, la qualité de service doit être ainsi intégrée directement dans les différents composants de ce dernier.

4 Système de stockage

Le système de stockage constitue sans doute la partie fondamentale d'un serveur vidéo puisqu'il est à la base de l'accès et de l'envoi des données vidéo. Cette partie a fait l'objet de recherches intenses. En effet, la majeure partie de la littérature actuellement disponible dans le domaine des serveurs vidéo, est consacrée aux systèmes de stockage. Cet engouement est justifié par la différence fondamentale qui existe entre les données statiques (texte, etc.) et les données continues (audio, vidéo) en termes de volume et d'isochronisme. Le principal objectif de ces travaux réside dans le fait de toujours satisfaire la contrainte d'isochronisme des données vidéo. Ces études peuvent être regroupées en quatre parties. La première partie se focalise sur les techniques de placement des données dans le système de stockage. La deuxième partie traite de l'ordonnancement des accès. La troisième partie regroupe les techniques, proposées dans le contexte des données isochrones, pour la tolérance et le recouvrement des pannes. Enfin, une partie des travaux traite des approches pour l'intégration des systèmes de stockage tertiaire (juke-box, *tape-robots*, etc.) aux serveurs vidéo. Ces parties sont présentées dans les paragraphes suivants.

4.1 Placement des données

Un système de stockage est généralement constitué de plusieurs disques. Il existe deux paradigmes de stockage d'une vidéo: (a) la vidéo est entièrement stockée sur un seul disque ou (b) elle est distribuée sur plusieurs disques. Le premier paradigme, s'il reste simple à gérer, n'offre pas en revanche un bon équilibrage de charge entre les différents disques. En effet, il suffit qu'une vidéo soit accédée plus fréquemment que les autres, ce qui survient souvent dans les applications vidéo, pour que ce disque devienne un goulot d'étranglement pour le système. Pour pallier cet inconvénient, d'autres approches proposent de dupliquer les vidéo les plus

"chaudes" sur plusieurs disques afin d'éviter la saturation d'un seul disque. Cette approche présente deux inconvénients majeurs: (a) l'espace requis pour la duplication est très important compte tenu du volume que représentent les données vidéo ; (b) il n'est pas toujours aisé de prédire à l'avance les fréquences d'accès aux données.

Le deuxième paradigme repose sur une fragmentation en un ensemble de blocs appelés *blocs de données* (BD). Ces blocs sont ensuite distribués sur les différents disques (on parle de *"stripping"*). Une politique d'allocation en tourniquet (*"Round-Robin"*) est souvent utilisée, dans la mesure où elle permet d'équilibrer la charge sur l'ensemble des disques sur lesquels la vidéo est stockée. Cette technique présente cependant l'inconvénient d'une complexité de gestion plus grande. D'autres schémas de distribution, comme l'allocation aléatoire utilisée dans RIO (voir prototype RIO, section 8.3.2) ou des variantes du tourniquet, ont été proposés dans la littérature [CL96].

Généralement, les BD sont de taille fixe, cependant, d'autres techniques exploitent la variabilité des débits des données vidéo (VBR) pour procéder à des fragmentations et à des placements plus fins. Dans ces techniques, les BD sont égaux, non pas en taille, mais en durée, chaque BD contient l'équivalent d'une seconde de vidéo par exemple. Ce découpage permet donc un meilleur contrôle de l'isochronisme des flux mais, reste difficile à mettre en œuvre [CZ96].

D'autre part, pour tenir compte de la géométrie variable d'un disque, Ghandeharizadeh *et al.* [CZ96] proposent d'exploiter une technique basée sur la notion de *multi-zoning* pour augmenter la capacité du disque à supporter plus de flux simultanés (voir la description du prototype Mitra, section 8.3.1).

4.2 Ordonnancement des accès disques

En présence de lectures concurrentes (plusieurs flux), il est primordial d'ordonnancer, de manière aussi optimale que possible, les accès aux disques.

Accès à un seul disque

Traditionnellement, l'ordonnancement des accès à un disque se base sur une des trois techniques suivantes: SCAN, RR (*Round-Robin*) et EDF (*Earliest Deadline First*). Dans la technique SCAN, les bras du disque parcourent les pistes en lisant les données demandées au moment où ils passent sur les pistes concernées. La technique RR sert les requêtes de lecture périodiquement selon un ordre défini. Finalement, plus orientée temps réel, la technique EDF sert les requêtes de lecture selon une priorité basée sur la date limite associée à chaque requête.

Dans le contexte de l'accès à des données continues, l'objectif des techniques d'ordonnancement sur un disque est de **minimiser à la fois le temps de lecture mais aussi le temps maximum entre deux lectures consécutives**. En effet, plus le temps entre deux lectures consécutives est grand, plus il faut lire un volume important de données à chaque cycle ce qui implique une utilisation mémoire plus importante.

Une multitude d'approches, combinant les techniques précédentes, ont été proposées. Reddy et al. [RW94] proposent par exemple un algorithme appelé SCAN-EDF. Dans cet algorithme, les accès sont basés, en premier, sur la priorité (EDF), ce qui permet de garantir les temps critiques des lectures. S'il arrive que plusieurs accès aient la même priorité, alors il sont servis en SCAN, réduisant ainsi les temps de latence.

Yu et al. [YCK93], proposent une technique appelée GSS (*"Grouped Sweeping Scheme"*) qui combine les deux algorithmes SCAN et RR. L'idée est la suivante: bien qu'il permette de maximiser le débit du disque, l'ordonnancement SCAN impose un temps maximum entre deux lectures consécutives pour un flux donné qui est relativement important (presque deux cycles). A l'inverse, l'ordonnancement RR permet de réduire ce temps (un seul cycle) mais ne maximise pas le débit du disque, d'où l'intérêt du GSS qui tire profit des avantages des deux approches en se basant sur la notion d'ensemble ordonnés. Dans cet algorithme, un cycle est découpé en ensembles ordonnés. On assigne à chaque flux un ensemble ordonné. L'ordonnancement SCAN est utilisé alors pour réduire les temps de recherche à l'intérieur de chaque ensemble ordonné. Les ensembles ordonnés sont servis en RR. Ainsi, si un seul ensemble est utilisé dans chaque cycle, le GSS se réduit à un ordonnancement SCAN. Si un seul flux est assigné à un ensemble ordonné, le GSS se réduit à un ordonnancement RR. La figure 5 illustre les caractéristiques en terme de temps de cycle et temps maximum entre deux lectures, des algorithmes SCAN, RR et GSS.

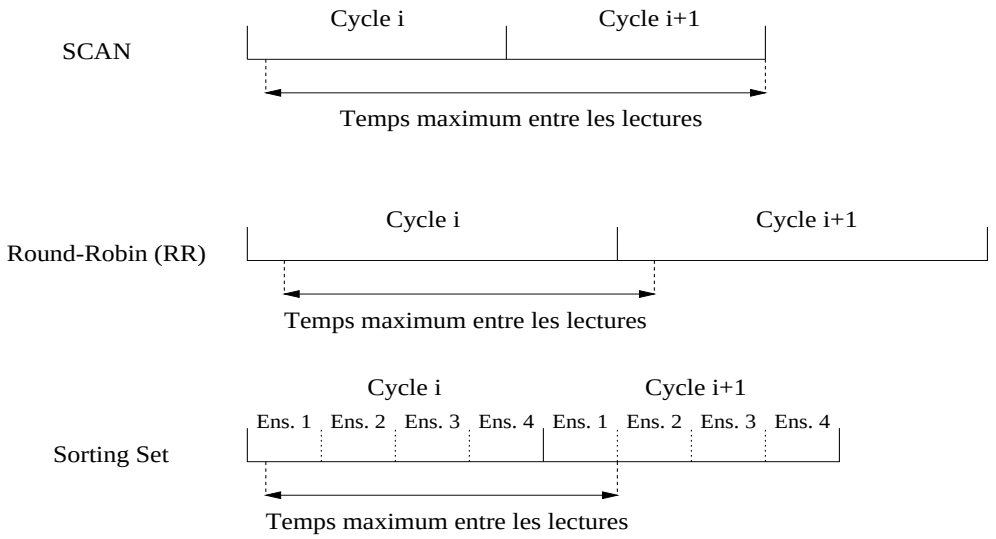

FIG. 5 – *Différentes politiques d'ordonnancement de lecture sur un disque.*

Accès à plusieurs disques

Comme mentionné précédemment, afin de s'affranchir de la limitation de la bande passante d'un seul disque et d'assurer un bon équilibrage de charge, les vidéos sont souvent découpées et distribuées sur plusieurs disques. L'objectif premier de l'ordonnancement des accès est alors de tirer profit de la nature périodique des données continues pour servir un maximum de flux simultanés.

Vidéo	Disques			
A	0	1	2	3
B	1	3	0	2
C	2	0	3	1
D	3	2	1	0
E	2	1	0	3

FIG. 6 – *Exemple de distribution cyclique de 5 vidéos sur 4 disques.*

Considérons l'exemple suivant [Red96]: supposons que la période de cycle, T, est divisée en un ensemble de 3 *slots*. Chaque *slot* correspond à la lecture et au transfert d'un bloc d'un disque vers la mémoire. Supposons également que le système contient 5 vidéos (A, B, C, D, et E) distribuées sur 4 disques (0, 1, 2, 3), cf. figure 6. Par exemple, la vidéo E est distribuée **cycliquement** sur les disques 2, 1, 0 et 3 dans l'ordre. Un ordonnancement possible pour la lecture de la vidéo E est illustré dans la figure 7 où les colonnes représentent les *slots* et la ligne représente l'ordre de lecture sur les disques. Ainsi, au *slot* 0 la lecture est lancée sur le disque 2 (E.2), au *slot* 3 la lecture est lancée sur le disque 1 (E.1), etc. Notons que la distance entre deux lectures consécutives est fixe due à la consommation continue des données. La figure 8 montre un ordonnancement des 4 flux simultanés des vidéos E, C, B et E (requêtes 0, 1, 2 et 3 respectivement). La principale contrainte qu'il faudrait respecter lors de l'ordonnancement des accès est que chaque *slot* ne doit contenir qu'une seule lecture sur un disque donné sinon, il y a un conflit de lecture. Pour cette raison, lors de l'ordonnancement de la requête 3 (lecture de la vidéo E), la première lecture a été placée au *slot* 2 et non pas avant car il y aurait eu conflit sur le disque 2 (E.2 au *slot* 0 et C.2 au *slot* 1).

Cet exemple montre le type de contraintes à respecter lors de l'ordonnancement des lectures sur plusieurs disques en parallèle. A partir de ce principe, une multitude de techniques d'ordonnancement ont été développées. Toutes ces techniques sont liées à la distribution utilisée (*stripping* total, sur tous les disques, *stripping* partiel, sur un sous-ensemble des disques, *stripping* à grain fin, *stripping* à gros grain, etc.).

Slot	0	1	2	3	4	5	6	7	8	9	10	11
Ordre de lecture	E.2			E.1			E.0			E.3		

FIG. 7 – *Exemple d'ordonnancement de la lecture de la vidéo E distribuée sur quatre disques (2,1,0 et 3).*

Requêtes	*Slot* 0	1	2	3	4	5	6	7	8	9	10	11
$R_0(E)$	E.2			E.1			E.0			E.3		
$R_1(C)$		C.2			C.0			C.3			C.1	
$R_2(B)$		B.1			B.3			B.0			B.2	
$R_3(E)$			E.2			E.1			E.0			E.3

FIG. 8 – *Exemple d'ordonnancement de quatre flux simultanés. $R_i(F)$ représente une requête n° i accédant le flux F. $F.i$ signifie la lecture de F sur le disque i. La contrainte à respecter est qu'il n'y ait pas deux lectures au même disque ordonnancées dans le même* slot.

4.3 Tolérance aux pannes

Comme nous venons de le voir, les applications vidéos ont une activité d'entrée/sortie intense. Or, le temps moyen de fonctionnement d'un disque sans panne[5] (MTBF) est estimé actuellement[6] à environ 1.000.000 heures. Si l'on suppose qu'un serveur contient 500 disques, hypothèse qui parait proportionnelle aux volumes des données vidéo, son MTBF est estimé à 5000 heures c'est-à-dire moins de 7 mois ! Il est donc primordial pour un système vidéo d'assurer son fonctionnement même en cas de panne d'un ou de plusieurs disques. Pour pallier ce problème, la redondance des données est nécessaire [GLP98].

On trouve deux paradigmes de redondance des données dans la littérature: (1) utilisation de données miroirs (*mirroring*) [BG88] et (b) construction d'information de parité [PGS88]. Les techniques basées sur le premier paradigme garantissent un recouvrement instantané en cas de panne d'un disque puisqu'il suffit de rediriger les accès vers les données miroirs. En contrepartie, l'espace nécessaire pour garantir ce recouvrement est très important. A l'opposé, les techniques basées sur la parité requièrent un espace de stockage moindre, mais, en revanche, nécessitent un temps CPU pour la construction des données perdues. De ces deux paradigmes, différentes techniques ont été dérivées et adaptées aux besoins des applications vidéo.

Disques miroirs

Dans cette approche, tout bloc de données dans le système de stockage est dupliqué. Cette approche induit un coût de stockage supplémentaire de 100 % si toutes les données sont dupliquées. La duplication peut être appliquée à deux niveaux: (a) au niveau du disque ou (b) au niveau du bloc de données.

Dans la première approche, le contenu du disque à dupliquer est entièrement recopié sur un disque miroir. Lorsque le disque courant tombe en panne, tous les accès sont redirigés vers le disque miroir. Cette approche, si elle a l'avantage d'assurer un recouvrement rapide, réduit la bande passante totale du système de moitié puisqu'en fonctionnement normal, la bande passante des disques miroirs n'est pas utilisée.

La duplication au niveau des blocs résout ce problème en distribuant les blocs miroirs d'un disque sur l'ensemble des disques restants. De cette façon, tous les disques du système sont utilisés, augmentant ainsi la bande passante globale. Dans le cas d'une panne d'un disque, les accès aux blocs sur ce disque sont redirigés vers les autres disques. Cependant, cette approche nécessite le maintien et la gestion de méta-données décrivant la localisation des blocs sur les différents disques. De plus, elle ne tolère pas la panne de plus d'un disque à la fois. Une solution pour ce problème consiste à distribuer les blocs miroirs sur un ensemble restreint de disques du système permettant ainsi au système de tolérer une panne par ensemble de disques.

5. *Mean Time Before Failure (MTBF).*

6. Un disque Hawk 1LP a un MTBF de 500.000 heures alors qu'un disque Barracuda 4LP possède un MTBF de 1.000.000 heures [Zim98]. Il y a quelques années, le MTBF d'un disque était de 300.000 heures [zRS95].

Disques de parité

Au lieu de dupliquer les données, les techniques à base de parité stockent uniquement des informations permettant, par la suite, de reconstruire les données perdues. Dans cette approche, les disques du système sont partitionnés en *groupes de parité*. Les blocs d'un fichier sont répartis sur les disques d'un groupe précis. Pour chaque groupe, les blocs de parité sont calculés en appliquant un opérateur tel qu'un XOR entre les blocs du groupe. Ainsi, sur un espace de stockage découpé en groupes comportant E disques, les données de parité peuvent être stockées en rajoutant un disque par groupe, soit une augmentation de l'espace de stockage d'un rapport $1/E$. Cette approche ne tolère la panne que d'un disque par groupe.

Dans le cas où les groupes de parité sont totalement indépendants, c'est-à-dire ne partageant pas de disques, la panne d'un disque risque de surcharger tout le groupe. En effet, afin de reconstruire les blocs perdus, tous les disques du groupe doivent être accédés, ce qui conduit à un goulot d'étranglement du système au niveau de ce groupe. Une solution pour pallier ce problème est basée sur le recouvrement cyclique des groupes de parité. Cette technique permet d'équilibrer les charges en cas de panne d'un disque sur les disques des groupes auxquels il appartient (voir figure 9). En revanche, elle tolère moins de pannes (pour l'exemple de la figure 9, un disque sur 3 groupes de parité).

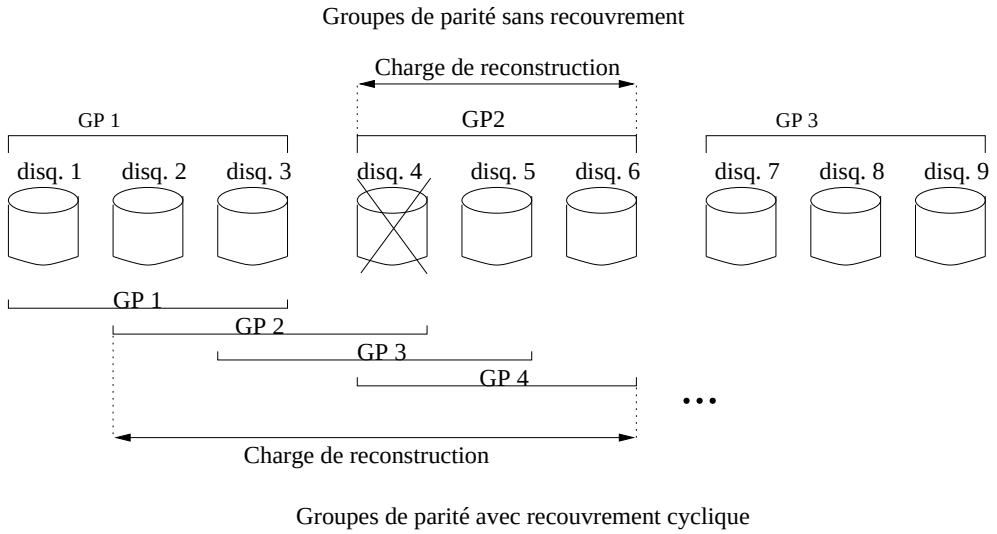

FIG. 9 – *Exemple de groupes de parité sans recouvrement et avec recouvrement cyclique.*

4.4 Stockage tertiaire

Vu le volume important des données impliquées dans les applications vidéo, le stockage tertiaire est souvent utilisé [Che98, KDST95]. Plusieurs applications comme la gestion des archives audiovisuelles, l'édition vidéo, la télévision numérique, etc. utilisent ce genre de stockage. L'intégration des systèmes de stockage tertiaire dans les serveurs vidéo a débuté avec les bandes magnétiques et les *tape-robots* [Che98]. En effet, ce genre de système constitue le prolongement naturel des magnétoscopes analogiques. L'avantage principal des systèmes de stockage tertiaire est d'une part leur grande capacité de stockage (de l'ordre des téra-octets) et d'autre part, leur faible coût, un Mo sur système tertiaire coûte beaucoup moins cher que sur disque ou encore en mémoire. Cependant, ils souffrent d'une lenteur d'accès significative et de débits nettement inférieurs à ceux des disques.

Depuis quelques années, d'autres périphériques ont fait leur apparition comme les disques optiques, combinés en juke-boxes, ou les DVD (*Digital Video Device*), rassemblés en batterie de DVD. Ces nouveaux systèmes présentent un rapport performance (capacité de stockage et rapidité d'accès)/prix plus intéressant et leur utilisation ne cesse de croître. Pour intégrer ces systèmes dans les serveurs vidéo, il est cependant nécessaire de faire cohabiter une hiérarchie de stockage et de développer des techniques adéquates pour

supporter la contrainte de continuité des flux vidéo. Nous ne ferons pas état, dans ce document, de ces techniques. Le lecteur pourra se référer à [Pan97, TP97, BR96, GDS95].

5 Gestion de la mémoire

Dans la section précédente, nous avons analysé la structure du système de stockage qui constitue la partie inférieure du serveur vidéo. Les données lues par le système de stockage sont d'abord momentanément stockées dans la mémoire avant d'être envoyées au gestionnaire d'interface. La tâche du gestionnaire de mémoire est de gérer l'espace mémoire en tenant compte de la nature isochrone des données vidéo.

Le volume important des données audiovisuelles ainsi que la garantie de flux continus imposent le développement de techniques de gestion de mémoire adaptées. Le problème de la gestion de la mémoire dans les serveurs vidéo peut être subdivisé en deux parties. La première concerne directement les politiques de gestion proprement dites (algorithmes de chargement et de remplacement). La deuxième partie se focalise sur les possibilités d'optimisation par le développement, principalement, d'heuristiques de partage, de cache et de pré-chargement. La frontière entre les deux parties est souvent floue et une majorité des techniques proposées dans la littérature sont des techniques intégrées (techniques de gestion et d'optimisation à la fois).

La gestion de la mémoire a été un domaine fertile en recherche essentiellement dans les systèmes de bases de données. Beaucoup de techniques ont été proposées dans la littérature. Ces techniques peuvent être regroupées en deux catégories: (a) les stratégies basées sur des heuristiques dynamiques et (b) les stratégies basées sur l'utilisation de la sémantique des applications. La première catégorie regroupe toutes les techniques classiques comme FIFO (*First In First Out*), LRU (*Least Recently Used*), LFU (*Least frequently Used*), etc. ainsi que les nombreuses techniques dérivées. La deuxième catégorie contient les techniques spécifiques à des applications particulières comme les systèmes d'aide à la décision.

Toutefois, la majorité de ces techniques n'a pas été développée pour gérer explicitement des flux continus de données vidéo. Ainsi leur utilisation peut, dans certains cas, ne générer aucun gain de performance. Considérons par exemple les stratégies de type LRU. Supposons qu'un serveur vidéo possède une mémoire d'une taille de 20 blocs et qu'il reçoive une requête pour une vidéo contenant 25 blocs mémoire (les blocs sont numérotés de 1 jusqu'à 25). Considérons un sens de flux ascendant: le bloc 1 est envoyé avant le bloc 2, etc. Une fois la requête servie, les blocs 6 à 25 occupent la mémoire du serveur. Si une requête accédant les blocs 5 à 15, arrive au serveur, le bloc 5 étant absent de la mémoire du serveur, le gestionnaire de mémoire doit donc le charger dans la mémoire. Pour ce faire, s'il applique une stratégie LRU, il va choisir le bloc 6 comme candidat au remplacement. Or c'est précisément le bloc 6 qui sera le prochain bloc demandé. Le candidat pour le remplacement sera le bloc 7 (selon LRU toujours) qu'il faudrait charger à son tour tout de suite après et ainsi de suite. Le bloc qui sera chargé au cycle prochain est celui qui est victime du remplacement au cycle courant ! Ce type de comportement montre clairement l'inefficacité des stratégies classiques et la nécessité du développement de techniques adaptées aux données vidéo.

Dans la littérature, une multitude de techniques de gestion mémoire adaptée aux données vidéo ont été proposées [RZ95, zRS96, TP97]. Nous ne ferons pas, dans ce document, une présentation détaillée, mais proposons de décrire les plus représentatives.

5.1 Stratégies de remplacement

Avant de présenter les différentes stratégies de remplacement, précisons tout d'abord nos hypothèses de travail. Nous reprenons les notations précédemment définies: le serveur vidéo dispose d'un *tas de mémoire* divisé en blocs égaux, l'unité d'échange entre le système de stockage et la mémoire est le bloc b, et sa durée de cycle est T. On suppose également que les données vidéo ont un débit constant (exprimé en nombre de blocs par seconde). Rappelons que l'objectif d'une stratégie de remplacement est, une fois la mémoire pleine, de choisir de manière optimale les blocs qui seront remplacés par les blocs objets de la requête courante. Généralement, la métrique de mesure de performance pour ce genre de stratégies, appelée *Miss Ratio* (MR), est le ratio entre le nombre total de défauts de blocs (blocs non trouvés dans la mémoire) et le nombre total d'accès aux blocs.

Stratégie BASIC

Dans [zRS96], Ozden *et al.* présentent une technique de remplacement, nommée *BASIC*, qui utilise la progression des flux dans le serveur à un instant donné comme critère de choix. L'idée de base est la suivante: étant donné un certain nombre de flux dans le serveur, en faisant l'hypothèse qu'il n'y a pas d'interaction avec les clients (progression normale des flux), les blocs qui seront accédés ultérieurement sont connus *a priori* du fait des débits constants des flux. On peut donc assigner à chaque bloc du tas un temps de *référence*. Ce temps est calculé en fonction de la position du bloc dans le flux. Par exemple, si on considère que la taille d'un bloc mémoire b est égale à 32 $ko = 0.256$ $Mbits$, le temps du dixième bloc dans un flux ayant un débit de 1.5 Mb/s est alors de $\frac{0.256 \times 9}{1.5} = 1.536$ secondes. L'algorithme sélectionne donc les blocs ayant le plus long temps de référence comme blocs candidats au remplacement. Il considère d'abord les blocs n'appartenant pas aux flux en cours puis le reste. Comparée à LRU et MRU, cette technique améliore le MR d'environ 30 %. Par contre, elle introduit des temps de calcul plus longs. A titre d'exemple, avec un tas de 2400 blocs, pour un cycle d'une seconde, BASIC prend 37 ms de calcul alors que LRU ou MRU ne prennent que 1 ou 2 ms pour le cas étudié dans [zRS96]. En outre, la technique BASIC n'est performante que sous l'hypothèse que les clients n'utilisent pas de fonctionnalités du type avance rapide ou recul rapide. Cette stratégie est souhaitable pour des applications VOD.

Stratégie L/MRP

Les besoins des applications plus interactives, en terme de gestion de la mémoire, sont différents des applications moins interactives. L'interactivité est en effet souvent gérée au niveau de la mémoire. Il est donc impératif de proposer des techniques de chargement et de remplacement intégrant de *facto* les fonctions d'interaction. Nous présentons une technique proposée par Moser *et al.* [MKK95] nommée L/MRP (*Least/Most Relevant for Presentation*). Cette stratégie modélise et intègre l'interactivité dans la gestion de la mémoire. Elle est basée sur le partitionnement des blocs mémoire en ensembles d'interaction. On assigne à chaque bloc une *valeur de pertinence* au remplacement ou au pré-chargement selon l'ensemble d'interaction auquel il appartient. Ainsi, les blocs avec une grande valeur de pertinence sont pré-chargés dans la mémoire alors que les blocs avec une faible valeur de pertinence sont sélectionnés pour être victimes du remplacement. Cette technique est dite intégrée puisqu'elle combine à la fois le remplacement et le pré-chargement des blocs mémoire. En se basant sur la définition des ensembles d'interaction, des scénarios d'interaction avec les clients peuvent être définis. Il s'agit dans ce cas, de **définir les valeur de pertinence pour chaque ensemble d'interaction**. Illustrons le principe de base de cette technique par l'exemple de la figure 10.

Une présentation, à un instant donné, est caractérisée par son point courant, qui correspond au bloc en cours de lecture, par son sens de progression qu'on appelle **sens de la présentation** et par la valeur d'avancement dans le flux qui correspond au nombre de blocs *"sautés"* lors de la progression (voir figure 10).

Dans l'exemple, trois ensembles d'interaction ont été définis: *History*, *Referenced* et *Skip*. L'ensemble *History* correspond aux blocs ayant déjà été envoyés au client et qui peuvent être potentiellement référencés dans le cas d'un retour en arrière par exemple. Les blocs proches du point de la présentation ont donc une pertinence plus élevée que ceux qui en sont loin. L'ensemble *Referenced* comprend tous les blocs qui seront lus et envoyés au client en se basant sur les caractéristiques de la présentation à un instant donné. Enfin, l'ensemble *Skip* regroupe les blocs qui seront *"sautés"* au vu des caractéristiques de la présentation.

Suite à une interaction précédente, on suppose que les blocs dont le numéro est en gras dans la figure ont été chargés dans la mémoire. Soit une présentation ayant les caractéristiques suivantes: à un instant donné, la présentation est au point 508, son sens est ascendant (suivant le nombre croissant des numéros de blocs), et sa valeur d'avancement (skip) est de +2 (voir figure 10). Les blocs les moins pertinents pour cette présentation sont dans l'ordre: 500, 501, 502, 503, 504, 519, etc. Par conséquent, les blocs candidats au remplacement sont: 501, 519, etc. Les blocs les plus pertinents à la présentation sont: 508, 510, 512, 514, etc. Par conséquent les prochains blocs pré-chargés seront: 512, 514, etc.

En attribuant une fonction de calcul de la pertinence pour chaque ensemble, on peut donc caractériser une application. Dans l'exemple précédent, les blocs de l'ensemble *History* ont une pertinence ascendante linéaire ; les blocs de l'ensemble *Referenced* ont une pertinence descendante parabolique, etc.

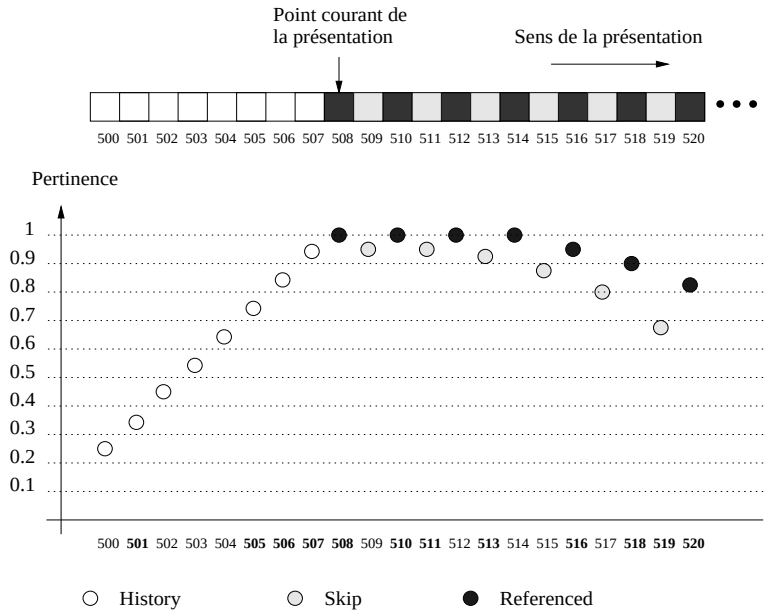

FIG. 10 – *L/MRP: exemple d'ensembles d'interaction et valeurs de pertinence associées aux blocs, les blocs dont les numéros sont en gras, sont chargé dans la mémoire.*

Stratégie de recouvrement de flux

Les stratégies de gestion de la mémoire présentées ci-dessus concernent seulement un flux indépendamment des autres flux dans le serveur. En d'autres termes, chaque requête se voit réserver un espace mémoire fixe déterminé lors de son acceptation dans le serveur qui demeure inchangé jusqu'à sa fin. Or, compte tenu de la progression constante des flux audiovisuels, il serait profitable de faire partager l'ensemble de l'espace mémoire entre les requêtes au lieu de faire des réservations à l'avance. L'idée de base de la technique présentée ci-dessous est de profiter de la dimension temporelle des données vidéo pour minimiser l'espace mémoire occupé [NY94]. Ainsi, cette technique permet de réduire de presque la moitié la taille de la mémoire nécessaire pour supporter n flux simultanés en les entrelaçant. Nous illustrons le principe de fonctionnement dans la figure 11.

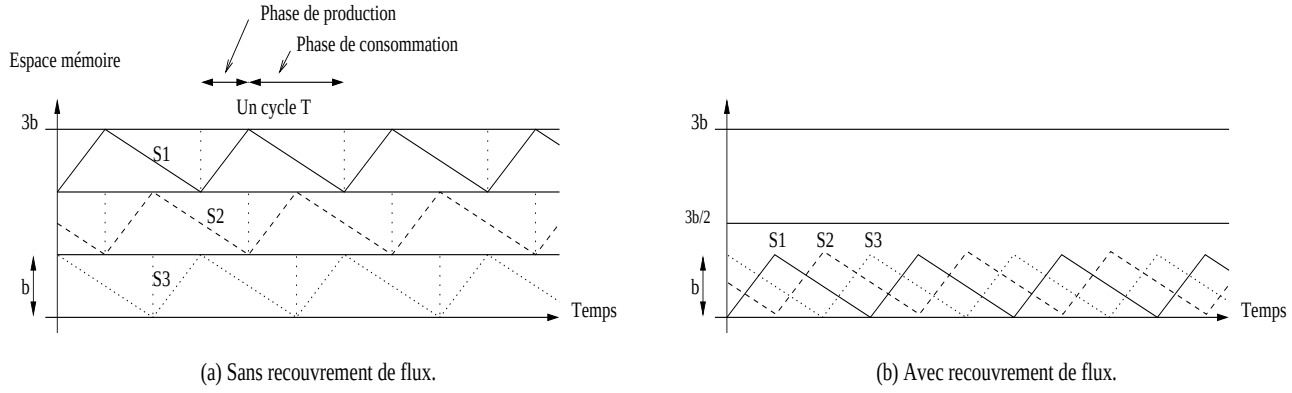

FIG. 11 – *Principe de recouvrement des flux continus.*

La condition 1, énoncée dans la section 3.1 (contrôle d'admission), stipule que la taille mémoire nécessaire pour supporter n flux vidéo est égale à la somme des tailles mémoire nécessaires pour chaque flux pendant un cycle. Or, dans un cycle du serveur, on distingue, pour chaque flux, deux phases: (1) la phase de chargement (production) durant laquelle la mémoire allouée à chaque flux est remplie, (2) la phase de consommation

durant laquelle les données sont consommées à des débits constants. La durée de chaque phase est déterminée en fonction du débit du système de stockage et du débit de consommation des données. Á titre d'exemple, considérons trois flux S_1, S_2 et S_3 de débit constant et égal et supposons que la phase de production est deux fois moins longue que la phase de consommation (c'est à dire elle est égale à $T/3$). La figure 11 (a) montre la progression des trois flux chacun dans l'espace mémoire qui lui est réservé. Or, connaissant la progression des flux, il est possible de *synchroniser* la consommation et la production des données de façon à ce que le même espace serve pour plusieurs flux. Ainsi, pendant la phase de consommation d'un flux, le serveur peut lancer la phase de production d'un autre flux dans l'espace mémoire libéré. C'est ce qu'on appelle un recouvrement des flux. La figure 11(b) montre le recouvrement des trois flux précédents. Ainsi, en recouvrant ces trois flux, l'espace mémoire total utilisé ($3b/2$) est inférieur de moitié à l'espace utilisé sans synchronisation ($3b$).

Cependant, cette technique impose que les flux gérés aient les mêmes débits. De même, les fonctions d'interaction sont gérées en dehors du serveur.

5.2 Optimisation de la gestion mémoire

Les stratégies de gestion de la mémoire présentées ci-dessus considèrent les flux indépendamment les uns des autres dans le sens où elles font abstraction des données accédées. Or, comme nous l'avons déjà mentionné, plusieurs requêtes peuvent accéder au même flux. L'objectif est donc de faire partager ces mêmes données entre différentes requêtes, de façon à minimiser les accès au système de stockage qui sont très coûteux en terme de temps de latence. De nombreux travaux de recherche se sont donc focalisés sur les possibilités d'optimisation de la gestion de la mémoire en se basant sur la notion de partage entre différents clients [KRT95, DS97, SG97]. Ces techniques, pour la majorité d'entre elles, sont basées sur l'idée de préserver (cacher) des blocs de mémoire pour une réutilisation ultérieure. Nous en présentons le principe en répondant à deux questions: (a) quelles sont les données à préserver? et (b) combien préserver? La réponse à la première question conduit à identifier, dans un contexte de flux continus, les blocs à préserver dans la mémoire. Quant à la deuxième question, elle est liée à la contrainte de la taille mémoire disponible à un instant donné.

En général, il est fréquent que plusieurs requêtes, accédant les mêmes données ou des données partagées, arrivent au serveur à des instants relativement proches. Ces requêtes, moyennant un cachage partiel des données, peuvent être servies par un seul flux, réduisant ainsi les accès au système de stockage. Ce cachage partiel repose sur la notion de *distance* entre les différentes requêtes. Le principe est illustré dans la figure 12.

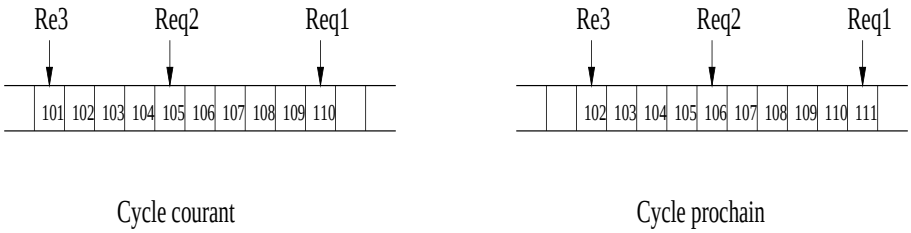

FIG. 12 – *Exemple de partage de flux.*

Dans l'exemple de la figure 12, trois requêtes concernent le même flux, $req1$, $req2$ et $req3$. Le système doit ainsi préserver dans la mémoire tous les blocs depuis le bloc lu par la requête $req1$ jusqu'au bloc courant lu par la requête $req3$. Le nombre de blocs mémoire entre les deux requêtes traduit une certaine *distance* entre elles. Par exemple, la distance entre la requête $req1$ et $req2$ est de 5 blocs et la distance entre $req1$ et $req3$ est de 9 blocs. Ceci sous-entend, d'une part, qu'il faut une taille mémoire d'au moins 9 blocs pour faire partager le flux entre les trois requêtes et d'autre part, que l'interaction avec les clients n'interfére pas avec ce partage. Ces techniques ne sont donc bien adaptées qu'aux applications peu interactives comme la VOD.

Si les stratégies de partage optimisent les accès au système de stockage, elles nécessitent en revanche plus de mémoire afin de préserver les données intermédiaires. Or, étant donné le volume important des données vidéo, l'espace mémoire nécessaire pour cacher des données peut rapidement devenir un goulot d'étranglement diminuant ainsi les performances du serveur. A titre d'exemple, considérons le serveur de la figure 13. On suppose que le serveur peut supporter huit flux simultanés accédant les données A, B, C, D, E et F (sans utiliser de partage). On constate qu'avec un partage, le serveur ne peut supporter que quatre flux (deux

flux sur A et deux flux sur B). Le partage de flux a donc conduit d'une part à une sous-utilisation d'une partie des ressources du serveur, en l'occurence le système de stockage (qui peut supporter 8 flux simultanés) et d'autre part à une sur-utilisation d'une autre partie des ressources du serveur, la mémoire. Il est donc impératif de trouver un bon équilibre entre la taille mémoire disponible et les distances de partage autorisées. Dans la littérature, différentes méthodes basées sur la définition d'un seuil appelé *"seuil de partage"* ont été proposées [CGM97]. Ce seuil peut être basé sur des critères de charge du serveur en terme de taux d'arrivée de requêtes comme c'est le cas dans [KRT95] ou sur des critères économiques (prix de la mémoire *vs* prix du flux disque[7]) comme c'est le cas dans [SG97].

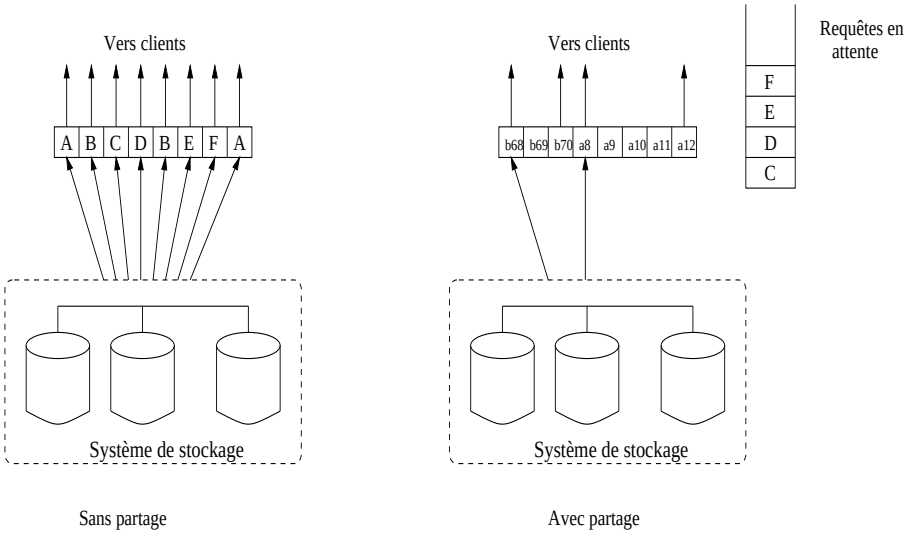

FIG. 13 – *Exemple de partage de flux inefficace.*

6 Gestionnaire d'interface

Ce composant est responsable de deux tâches principales: la transmission des blocs de données aux clients à travers le réseau de distribution qui les lie au serveur vidéo et la gestion des interactions avec les clients (support des fonctionnalités magnétoscope).

6.1 Support des fonctionnalités magnétoscope (VCR)

Pour le mode *client pull*, ces fonctionnalités sont gérées par le client qui dans le cas d'avance rapide va demander non pas le bloc consécutif à celui qu'il vient de recevoir mais un bloc plus lointain. Dans le cadre du mode *push*, les fonctionnalités de magnétoscope ont été intégrées directement, lors de la conception d'un serveur vidéo, dans les différentes composantes de ce dernier. On trouve un certain nombre de stratégies liées au système de stockage sous-jacent [CKY95], au placement des données [SV99], aux formats de compression utilisés [CK96, And96], etc. Par conséquent, chaque prototype possède ses propres techniques de support des fonctionnalités VCR. A titre d'exemple, dans le serveur Mitra (voir paragraphe 8.3.1) l'avance rapide et le recul rapide sont implantées via deux flux supplémentaires générés lors du stockage d'une vidéo. Ainsi, une requête d'avance rapide sur le flux normal entraînera le basculement du client sur le flux d'*"avance rapide"* jusqu'à la prochaine requête d'avance normale. De même pour le recul rapide.

6.2 Transmissions sur le réseau de distribution

Les réseaux actuels sont souvent optimisés pour le trafic de données fragmentables et non pour le trafic de flux continus. Lorsqu'il y a saturation du réseau, les flux transportant des données temps réel ne peuvent

7. Si on suppose qu'un disque coûte 1000 FF et qu'il peut supporter 10 flux simultanés, alors le coût d'un flux est de 100 FF.

plus être délivrés en continu. Depuis la forte évolution du multimédia, de nombreux travaux ont été faits dans le cadre notamment des groupes de travail de l'IETF [8] pour adapter les réseaux aux besoins des applications multimédia.

Transmissions multipoint (*multicast*). Dans un réseau de distribution, l'objectif est de réduire l'utilisation de la bande passante par chaque client. En effet, si pour chaque utilisateur on arrive à communiquer moins de données (via une compression par exemple), on pourra soit servir plus d'utilisateurs soit augmenter la qualité de service pour chaque utilisateur. Dans ce cadre, l'idée des communications multipoint propose, plutôt que d'envoyer un ensemble de paquets à chaque client (unicast) ou de diffuser les paquets à tous les clients (broadcast), d'envoyer un flux de paquets, une seule fois à un groupe de clients. Ainsi, une vidéo transmise en multicast à 4 clients nécessitera un seul envoi de la part du serveur, les routeurs se chargeant de transmettre les données aux clients ayant souscrit à cette transmission multipoint. Les algorithmes de routage multipoint actuellement utilisés (PIM [9], MOSPF [10]), sont basés sur l'adressage IP multipoint.

Protocoles spécifiques. Face aux lacunes de protocoles tels que TCP adaptés ni aux données temps réel ni à l'adressage multipoint, le protocole RTP (Real Time Protocol) a été développé. Ce protocole, indépendant du réseau et des couches de transport sous-jacentes est très flexible et peut donc s'adapter à différents types d'applications multimédia. Le principe est simple: avant d'être envoyées, les données sont mises en paquets au format RTP spécifique à leur encodage puis poussées sur le réseau via UDP. L'en-tête d'un paquet RTP contient des informations sur la synchronisation et l'affichage des données. Ceci permet de détecter d'éventuelles pertes de paquets ou une réception des paquets dans le désordre. En parallèle de RTP, on trouve le protocole RTCP qui s'occupe de la qualité des transmissions. Ainsi chaque client envoie régulièrement des paquets RTCP sur le réseau et analyse ensuite les réponses pour détecter des problèmes éventuels.

Outils de gestion de la qualité de service. Dans des communications réseaux classiques, la garantie de qualité de service est souvent très faible. La politique *Best Effort* est souvent adoptée par les routeurs qui mettent au même niveau les paquets temps réel et les autres. Pour pallier ce problème des outils (Intserv [11], RSVP [12], DiffServ [13]) permettant de contrôler la qualité de service ont été développés. Ils permettent aux applications, par exemple, de réserver dynamiquement des ressources réseaux.

7 Architecture matérielle

Comme mentionné précédemment, les applications vidéo sont nombreuses et variées. Cette variété impose donc des exigences diverses (puissance de calcul, capacité de stockage, etc.) au niveau du serveur vidéo. Par conséquent, les plates-formes matérielles, sur lesquelles les serveur vidéo sont implantés, varient, selon les besoins des applications, de simples machines monoprocesseur jusqu'aux ordinateurs massivement parallèles. Évidemment, chaque type d'architecture possède ses avantages et ses inconvénients. Les facteurs déterminant dans le choix d'une plate-forme matérielle sont, d'une part, la nature de l'application visée (VOD, traitement vidéo, édition vidéo, etc.) et d'autre part, son coût.

7.1 Architecture à base de machine SMP

Historiquement, les machines SMP (Symmetric Multi-Processors) ont été les premières à servir comme base au développement des serveurs vidéo. Ce choix était motivé par le fait qu'elles étaient les seules à posséder les ressources nécessaires à un serveur vidéo en termes de puissance de calcul, de taille mémoire (de l'ordre de plusieurs giga-octets) et de capacité de stockage (de l'ordre de plusieurs téra-octets) [JSCB95a, JSCB95b].

8. Engineering Task Force, `http://www.ietf.org`
9. `http://www.ietf.org/html.charters/pim-charter.html`
10. `http://www.ietf.org/html.charters/mospf-charter.html`
11. `http://www.ietf.org/html.charters/intserv-charter.html`
12. `http://www.ietf.org/html.charters/rsvp-charter.html`
13. `http://www.ietf.org/html.charters/diffserv-charter.html`

C'était donc un choix naturel de bâtir des serveurs vidéo autour de machines SMP. Ces dernières présentent les avantages suivants.

Meilleures performances. La puissance de calcul des machines SMP, qui reste nettement supérieure à celle des machines monoprocesseur, permet d'exécuter des tâches nécessitant un temps CPU important comme l'indexation ou l'analyse de données vidéo. De plus, les machines SMP sont capables de gérer des espaces de stockage importants nécessaires au stockage de vidéos.

Gestion centralisée et simplifiée des flux vidéo. L'unicité de la mémoire dans les machines SMP permet une gestion plus simple, à l'opposé des machines à mémoire distribuée, et conduit à des optimisations importantes comme le partage de flux entre clients.

Au fur et à mesure de l'émergence de nouvelles applications vidéo, des besoins nouveaux et surtout de nouvelles contraintes sont apparus. Si, au début des premiers prototypes de serveurs vidéo, la priorité était donnée à la fonctionnalité, c'est-à-dire assurer des flux vidéo continus, les priorités ont depuis changé. Ainsi, pour une application de gestion d'archives vidéo, la scalabilité[14] est un besoin vital que doivent prendre en compte les concepteurs du serveur vidéo lors du choix de la plate-forme matérielle cible. Une autre contrainte, qui est devenue déterminante ces dernières années, est le coût d'un serveur vidéo. Si, du point de vue des performances, les architectures à base de SMP ont de sérieux atouts pour elles, en revanche, du point de vue de la faisabilité économique, elles souffrent d'un rapport performance/prix nettement déséquilibré. On peut résumer les inconvénients des serveurs vidéo à base de SMP ainsi:

Mauvais rapport performance/prix. En effet, les machines SMP, à partir de la dizaine de processeurs, restent relativement chères de part leur complexité technologique.

Evolutivité complexe. Le passage de serveurs de petite taille à quelques processeurs vers des serveurs plus important nécessitant plusieurs dizaines voire centaines de processeurs, nécessite une remise en cause matérielle et logicielle de la conception du serveur.

Tolérance aux pannes. L'unicité de la machine ne permet pas de tolérer des pannes autres que les pannes de disques.

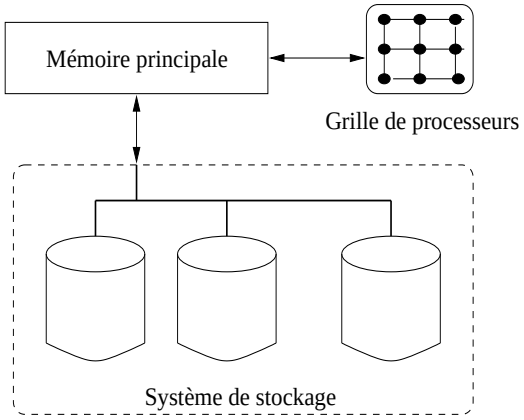

FIG. 14 – *Architecture à base d'une machine SMP.*

14. *Scale up*: capacité à maintenir un temps de réponse constant quand flux de requêtes et bases de documents augmentent. Cela implique l'ajout de composants matériels.

7.2 Architecture à base de grappe de machines

Les évolutions enregistrées, depuis quelques années, en matière de réseaux d'interconnexion et de puissance de calcul des machines monoprocesseur ont permis l'émergence d'une nouvelle classe de plates-formes matérielles capable de servir de base aux développements de serveurs vidéo. Le principe des architectures à base de grappe de machines[15] est de connecter un ensemble de machines, chacune possédant sa propre mémoire et son propre espace de stockage[16], via un réseau d'interconnexion à haut débit. Chaque machine connectée est appelée **nœud**. Ainsi, l'ensemble des nœuds, via des mécanismes de partage entre eux, constitue une seule machine logique. Les nœuds d'une grappe peuvent être divisés logiquement en deux catégories: les nœuds *d'interface* (*front-end nodes*) et les nœuds de *stockage* (*back-end nodes*). Les nœuds d'interface assurent la communication entre les clients et le serveur vidéo. Ils reçoivent les requêtes des clients, les analysent (contrôle d'admission), et les transmettent aux différents nœuds de stockage sur lesquels résident les données demandées. Les nœuds de stockage reçoivent les requêtes des nœuds d'interface, recherchent les données demandées et les transmettent aux nœuds d'interface qui, à leur tour, les transmettent aux clients. Évidemment, ce scénario d'exécution se déroule sous la contrainte d'isochronisme des données vidéo.

Selon l'attribution logique des nœuds de la grappe en nœuds d'interface ou nœuds de stockage, on distingue deux types d'architecture en grappe [TMDV96]. Dans la première, dite *"Two-Tier"*, les nœuds d'interface et les nœuds de stockage sont localisés sur des machines différentes. Les nœuds d'interface ne communiquent pas entre eux et la distribution des données sur les nœuds de stockage est assurée par un mécanisme de disques virtuellement partagés (voir figure 15). Dans la seconde architecture, dite *"Flat"*, tout nœud de la grappe sert à la fois de nœud d'interface et de nœud de stockage. Chaque nœud doit donc assurer simultanément les fonctions de communication du nœud d'interface et les fonction de stockage du nœud de stockage (voir figure 16).

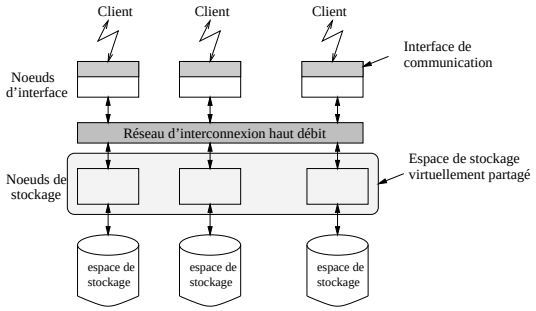

FIG. 15 – *Architecture en cluster (Two-Tier).*

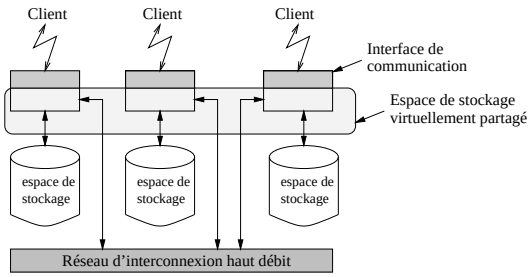

FIG. 16 – *Architecture en cluster (Flat).*

Les architectures en grappe présentent des avantages intéressants.

Une bonne scalabilité. En effet, les opérations d'extension se limitent seulement à la connexion d'une nouvelle machine à l'ensemble existant.

Un rapport performance/prix nettement favorable. Une grappe est composée d'éléments peu chers (simples PC, disques) en comparaison aux SMP. Ce point est un argument de taille quant aux développements futurs des serveurs vidéo.

Une bonne tolérance aux pannes. La distribution du système de stockage sur plusieurs nœuds facilite le développement de techniques efficaces de tolérance aux pannes via des mécanismes de duplication de données et de recouvrement.

15. *Cluster* en anglais.
16. Architecture *Shared nothing*.

Si les architectures en grappe présentent des avantages très intéressants, il en demeure néanmoins qu'elles souffrent de certains points faibles:

Coût de développement important. La mise en œuvre des mécanismes de collaboration entre les différentes machines interconnectées reste une tâche complexe et fastidieuse.

Gestion complexe. Maintenir la cohérence de méta-données dupliquées sur plusieurs nœuds est plus complexe que dans une architecture centralisée.

7.3 Configuration du serveur vidéo

Un autre aspect important, lors de la conception d'un serveur vidéo, est celui de l'évaluation de la configuration requise. Cette configuration est évidemment tributaire de l'application cible. En général, l'évaluation d'une configuration matérielle concerne essentiellement la détermination de la taille de la mémoire requise, de la bande passante disque nécessaire, des techniques de placement de données les plus adaptées, etc. Dans la littérature, un travail académique conséquent a porté sur la proposition d'approches permettant de déterminer ces paramètres. Ces travaux peuvent être regroupés, en fonction de l'architecture matérielle sous-jacente, en deux catégories: la première catégorie regroupe les travaux sur les architectures en grappe. On note les travaux de Tewari et al. [TMDV96] qui ont proposé un modèle analytique basé sur la théorie des files d'attente. La deuxième catégorie concerne les architectures à base de machines SMP [JSCB95a, JSCB95b].

Outre le choix de la plate-forme matérielle, les applications vidéo ont une incidence directe sur la détermination des paramètres de cette dernière. Par exemple, si une application nécessite des temps de réponse assez courts, le serveur vidéo lui correspondant aura certainement besoin d'un cache mémoire important. A l'opposé, si une application, comme la gestion d'archive vidéo, requiert un espace de stockage important, le serveur vidéo lui correspondant doit gérer, de la manière la plus efficace possible, une hiérarchie de stockage (stockage tertiaire sur des bandes ou des disques optiques). Ainsi, le problème de la conception d'un serveur vidéo se résume en général en deux étapes: (a) choix de la plate-forme matérielle (SMP, grappe de machines, etc.) et (b) détermination des paramètres de la plate-forme matérielle choisie. Ces paramètres peuvent être aussi bien *matériels*, comme la taille des mémoires, que *logiciels*, comme les techniques de placement, contrôle des données. Il en découle alors une forte corrélation entre le choix architectural de la plate-forme matérielle et les techniques de gestion sous-jacentes (gestion de la mémoire, techniques de recouvrement de pannes, etc.).

8 Prototypes

Dans cette partie, on dresse un panorama des principaux systèmes qui ont abouti aux avancées les plus significatives tant du point de vue théorique qu'applicatif. Ensuite, nous présentons deux prototypes commerciaux basés sur deux plates-formes matérielles différentes et trois prototypes de recherche couvrant les différentes facettes des développements futurs vers lesquels s'orientent de plus en plus les chercheurs. Chacun de ces prototypes possède un point fort particulier que nous essayons de mettre en relief.

Tiger. Ce serveur est construit sur une architecture distribuée (grappes de PC). Son originalité réside dans la gestion d'un gestionnaire de ressource distribué.

Fellini. Basé sur une mémoire partagée, ce serveur est bâtit sur une machine SMP. Il utilise des algorithmes de gestion de mémoire et une modélisation de la politique d'admission intéressants.

Mitra. Comme dans Tiger, Mitra est basé sur une architecture distribuée. Mitra donne de bonnes performances grâce à un important travail de bas niveau sur le placement des données sur un disque (Multi-zones).

RIO. RIO propose un nouveau type de placement des données, l'allocation aléatoire. Ce placement des données est particulièrement intéressant dans le cadre d'un serveur multimédia supportant plusieurs applications.

Symphony. Comme dans le cas de RIO, Symphony supporte plusieurs types de données multimédia. Toutefois, l'approche choisie est très différente de celle de RIO dans lequel on traite toutes les données de la même manière (optimisation globale). Dans Symphony, en revanche, on associe à chaque type de donnée des techniques de lecture ou d'écriture spécifiques (optimisation locale) tout en gardant un serveur unique.

Après un descriptif historique des développements des serveurs vidéo (paragraphe 8.1), le paragraphe 8.2 présente deux prototypes commerciaux, à savoir **Tiger** et **Fellini**. Dans la section 8, nous présentons trois prototypes académiques: **Mitra**, **RIO** et **Symphony**. Enfin, le dernier paragraphe 8.4 dresse un tableau comparatif des prototypes étudiés.

8.1 Historique

En une décennie, de nombreux serveurs vidéo, résultat de l'engouement qui a animé la communauté des chercheurs et des industriels pour les applications vidéo, ont été proposés dans la littérature.

Les premiers travaux, portant sur la gestion des données multimédia, commencent au milieu des années 80. Des systèmes tels que Diamond [TFC+85], Muse [GTF+87] ont été développés pour le stockage et l'échange de documents contenant des images. Un peu plus tard, vers la fin des années 80, de nouveaux travaux s'orientent vers les données audio. On mentionne, entre autre, le système de fichier Multimédia, développé par Sun [Sun89], qui stocke des données audio sous forme de fichiers Unix et propose un partage de ces données avec NFS ainsi que le serveur audio VOX [AHL+91].

Les travaux sur les données vidéo commencent à émerger au début des années 90, puis deviennent rapidement un domaine de recherche très actif. Tout d'abord, quelques investigations [Hop90, Mor90] ont été faites concernant des mécanismes de stockage de bas niveau, puis très vite de nouveaux thèmes de recherche sont abordés.

Á cette époque, les contraintes matérielles étaient plus pesantes qu'aujourd'hui. Les bandes passantes des disques et celles des réseaux étaient très largement inférieures. Notamment, les disques de grande capacité manquaient de fiabilité alors que les disques haut débit restaient très chers. Dans ce contexte, les petits disques ont représenté un bon compromis entre prix et performances. Certains travaux ont alors porté sur l'optimisation des accès aux disques. Patterson *et al.* [PGS88] ont proposé le système de RAID, permettant d'accéder à plusieurs disques en parallèle tout en procurant une certaine tolérance à la panne d'un disque. Quelques prototypes [Che90, C+93, L+92] ont été développés sur le principe des RAID. Toutefois, les résultats obtenus ne sont pas ceux escomptés au niveau du rapport performances/prix. En effet, un système RAID matériel reste très cher et les coûts de recopie de cache ou DMA limitent souvent les performances.

En parallèle à ces travaux, des études sont apparues portant sur l'exploitation de la nature continue des flux vidéo. Plus particulièrement, ces travaux se sont focalisés sur les techniques d'optimisation des accès aux disques [RV91, RV93, YCK93], incluant les algorithmes de placement des données et les techniques d'ordonnancement des accès. A partir de ces travaux, des prototypes ont vu le jour tels que Continuous Media File System [AOG92], Continuous Media Storage Server [LS93] ou Shark [Has93], le prototype d'IBM.

Les premiers prototypes de serveurs vidéo ont montré leur faisabilité technique. Ainsi, entre 1992 et 1995, la plupart des grand constructeurs de matériel informatique (IBM, SGI, HP) ou de logiciels (Oracle, Microsoft, IBM), les compagnies de téléphones ou câble (AT&T, Ameritech, Bell Atlantic, Time Warner, TCI) se sont lancées sur le marché de la vidéo à la demande ([DVV94], [LV94]). L'objectif était de permettre à des clients distants de recevoir de la vidéo (films, annonces, etc.), *via* Internet, tout en leur offrant une facilité d'utilisation comparable à celle d'un magnétoscope. Malheureusement, ces compagnies se retrouvèrent toutes face à un échec, ce qui engendra un certain découragement général. Cet échec s'explique essentiellement par les limitations d'Internet. En effet, Internet au dessus de lignes téléphoniques classiques, est encore trop limité en bande passante et oblige de câbler avec un autre réseau (ATM, Ethernet) les clients voulant accéder aux services du serveur vidéo. De ce fait, du point de vue du client, le coût de la connexion d'un serveur vidéo reste élevé (matériel + logiciel) et donc le simple fait de cliquer sur le film que l'on veut voir revient souvent plus cher que de descendre louer une cassette. Ainsi le rêve d'amener le serveur vidéo chez tout le monde s'écroule. Toutefois, cet échec reste partiel. En effet, un certain nombre de serveurs vidéo ont été utilisés dans le cadre de grosses infrastructures tels que des hôtels ou des hôpitaux câblés avec un réseau supportant les débits requis pour la vidéo. Parmi les serveurs vidéo commerciaux développés à cette époque, on trouve

Tiger [BB+96b] de Microsoft, Oracle Media Server [LOP94], Fellini [MNz+96] de AT&T ou le serveur SGI [NLO95] utilisé par Time Warner.

A partir de 1995, on assiste au développement de nombreux prototypes académiques résultant de la recherche sur un modèle ou un point d'implémentation précis. Certains ont repris la technique RAID en proposant un RAID logiciel appliqué aux noeuds d'une solution distribuée (Server Array [BB96a], RAIS [WL97]). D'autre part, on voit différentes architectures proposées pour construire un serveur vidéo (cf. section 7). Par exemple, les serveurs SPIFFI [FBW96], Mitra [GZS+97] sont construits à partir de composants standards du marché, leur objectif étant d'obtenir le meilleur rapport performance/coût possible. D'autres serveur tels que celui développé par l'université du Minnesota [HLL+95] ou la première implémentation du serveur RIO [MSB97] ont fait le choix de SMP, plus coûteux mais souvent plus performants au niveau des entrées sorties. Autre solution, le serveur MARS [BPC94] propose une implémentation basée sur une architecture massivement parallèle.

Avec l'évolution des réseaux d'interconnexion (Giga-Ethernet, ATM, Myrinet, etc.), la solution distribuée va très vite s'imposer au détriment de grosses machines SMP. Actuellement, nombreux sont les serveurs vidéo construits à base de grappes de PC connectés par ATM (Elvira [SLM97]), Fast-Ethernet (2ème implémentation de RIO [FSM98], Swarm [HMS99]) ou plus récemment utilisant Myrinet [CCIV99, SKKP99] ou le réseau SCI [VV99].

Depuis 1998, de nouveaux travaux de recherche sont menés, notamment sur l'approche d'intégration (cf. 1.3). Les serveurs Symphony [SGRV98] (cf. 8.3.3) ou RIO [MSB97] (cf. 8.3.2) ne sont donc pas dédiés uniquement à la vidéo mais à différents types de média. Les développeurs argumentent que, face à des applications multimédia telles que la visualisation 3D, l'approche de séparation ne donne pas des résultats optimaux.

8.2 Prototypes commerciaux

8.2.1 Tiger (Microsoft): une solution totalement distribuée.

Tiger [BB+96b] a été développé entre 1993 et 1994 par MicroSoft et est à la base de NetShow. Ce serveur vidéo est basé sur un ensemble de PC connectés par un réseau à commutation de paquets. L'objectif de Tiger est de supporter un grand nombre de flux vidéo tout en restant bon marché, scalable et procurant une forte disponibilité des données.

Les données manipulées dans Tiger sont distribuées sur les disques de manière globale et cyclique. Ceci permet d'assurer un bon équilibrage de charge. La tolérance aux pannes est assurée par un système de miroirs distribués de degré d: chaque bloc est découpé en d sous-blocs qui sont répartis sur plusieurs disques. Ainsi, même en présence de pannes, l'équilibrage de charge peut être maintenu. Tous les nœuds sont des *nœuds de stockage* à l'exception du nœud, appelé nœud contrôleur, qui est dédié aux connexions avec les clients.

Contrairement à beaucoup d'autres prototypes, dans Tiger, l'ordonnancement des flux sur les différents disques est fait de manière distribuée. Ainsi, lors de la lecture d'un flux, les différents nœudssur lesquels ces données sont stockées communiquent entre eux pour assurer l'isochronisme des lectures. En effet, à chaque flux est associé un *viewer state record (VSR)* qui contient des informations sur l'adresse du client, le fichier lu, la position dans le fichier, etc. Quand un nœud a effectué une lecture pour un flux donné, il envoie le VSR au nœud suivant et ainsi de suite en suivant la distribution cyclique des données. Ce mécanisme permet d'éviter que la panne du nœud responsable de l'ordonnancement global entraîne une panne générale du serveur. Dans ce schéma, l'équilibrage de charge est maintenu grâce à la distribution cyclique des données mais aussi par le fait que Tiger ne supporte que des flux CBR. Ainsi, tous les flux naviguent de disque en disque à la même vitesse. Lorsqu'on a atteint le nombre maximum de flux pour chaque disque, on ne peut plus accepter de nouveaux clients.

Toutefois, la contrainte du CBR n'est plus acceptable à l'heure où de nouveaux formats vidéo apparaissent et où le VBR se développe. Des études [BFD97] ont été faites pour supporter des flux VBR dans Tiger, mais ces solutions n'ont pas été implémentées. Du point de vue performances, Tiger peut supporter jusqu'à 68 flux simultanés à 6 Mbit/s pour 5 PC (Pentium 133 MHz, 48 Mo RAM) ayant chacun 3 disques de 2 Go et connecté par ATM.

8.2.2 Fellini (AT&T): une mémoire partagée

Fellini [MNz+96] est un serveur de stockage multimédia développé par AT&T. Ce serveur supporte des opérations temps réel et non temps réel et peut donc délivrer des flux vidéo. Fellini est un exemple de serveur basé sur une architecture à mémoire partagée (machines multiprocesseur ou tout simplement une station de travail avec beaucoup de mémoire et de disques).

D'un point de vue conceptuel, on peut distinguer deux aspects dans l'architecture de Fellini: le fonctionnement interne du serveur et l'interaction du serveur avec d'autres composants du système multimédia plus général.

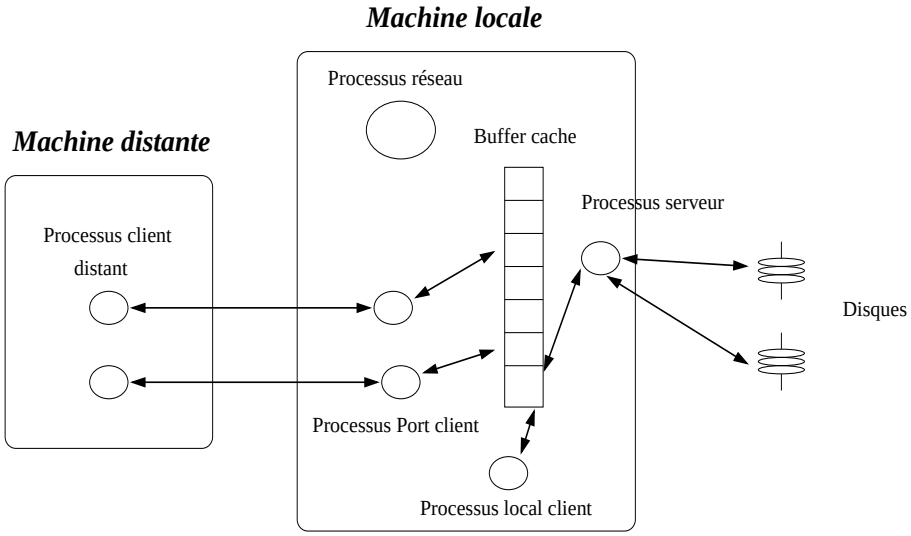

FIG. 17 – *Architecture globale de Fellini.*

La figure 17 montre une vue d'ensemble de l'architecture générale de Fellini. Le serveur est responsable du transfert des données entre le buffer servant de cache et le stockage secondaire (disques). Le *processus réseau* reçoit les demandes de connexion de la part des clients et leur associe alors un *processus port client* qui leur permet d'accéder aux données. Le *buffer cache* étant en mémoire partagée, tous les processus peuvent donc y accéder. Pour une opération de lecture, le fonctionnement est le suivant: le *processus local client* va lire les données dans le *buffer cache* puis les envoie au client par le *processus port client*. Dans le cas d'une écriture, le client envoie les données à stocker au *processus local client* qui écrit les données dans le *buffer cache*. Dans les deux cas, le *processus local serveur* gère le lien entre le *buffer cache* et les disques. Il va lire ou écrire sur les disques si nécessaire.

Le serveur lui-même est composé de trois agents: un contrôleur d'admission, un gestionnaire de cache et un gestionnaire de stockage. Le contrôleur d'admission a un rôle d'allocateur de ressources. Il alloue les ressources (bande passante des disques, taille des buffers) aux clients et assure que les contraintes temps réel des clients sont bien respectées. Le gestionnaire de cache gère le *buffer cache* afin d'optimiser la disponibilité des données. Enfin, le gestionnaire de stockage est responsable de la gestion des fichiers et de l'espace disque. Pour chacun de ces agents, des schémas d'optimisation de leur tâche ont été développés. Ainsi, pour obtenir une politique d'admission permettant d'accepter un maximum de clients tout en préservant une bonne qualité de service, une modélisation minutieuse des accès disques a été proposée. Elle est basée sur un ordonnancement par cycles des requêtes des clients et sur les performances disques (bande passante, temps de rotation, temps de recherche) dans les pires cas. Les équations ainsi obtenues permettent alors au *contrôleur d'admission* de décider de l'acceptation ou non d'un nouveau client.

Dans le cadre du gestionnaire de cache, un nouveau schéma de remplacement de bloc est proposé. Les politiques de remplacement classiques ne sont en effet pas adaptées aux spécificités continues des données multimédia telles que la vidéo. Fellini utilise donc sa propre politique de remplacement (voir [MNz+96] pour

plus de détails). Malheureusement, il n'y a pas de résultats de performance disponibles bien que ce serveur ait été implémenté sur plusieurs systèmes d'exploitation et commercialisé sous le nom de *CineBlitz* en 1997.

8.3 Prototypes académiques

8.3.1 Mitra (USC): optimisations des accès disques

Mitra [GZS+97] est un serveur vidéo développé par l'équipe bases de données de USC (University of Southern California). Le principal objectif est de construire un serveur vidéo avec des composants standard du marché et donc peu chers (simples machines possédant des disques et connectées avec un réseau). Ce serveur est le résultat d'études sur des techniques d'optimisation d'accès aux disques. La principale technique d'accès aux disques utilisée est le *multi-zoning* [GM98]. Cette technique est basée sur le fait qu'un disque possède plusieurs zones, chacune ayant un débit différent. Rappelons qu'un disque est composé d'un ensemble de plateaux. Sur chaque plateau on trouve un certain nombre de pistes. Ainsi la piste intérieure du disque contient moins de données que la piste extérieure. La fréquence de rotation du disque étant constante, la vitesse de déplacement de la tête de lecture diminue pour les pistes intérieures. De plus, comme la densité de stockage varie peu entre les différentes zones du disque, si la tête de lecture est sur la piste intérieure, en un tour, elle lira moins de données que si elle était sur la piste extérieure. Il y a donc une différence de débit entre les pistes [Zim98].

Concernant le contrôle d'admission, on cherche à évaluer le temps maximum d'une lecture sur un disque. La plupart des modèles analytiques des disques se basent alors sur le débit de la piste interne (le cas minimum). Ce schéma sous-utilise la bande passante disque. La technique d'optimisation d'accès aux disques développée dans Mitra est basée sur le regroupement de pistes en zones. Une zone est un ensemble de pistes ayant la même capacité de stockage. Les disques avec des zones ont été introduits par les fabriquants de disques pour augmenter leur capacité de stockage. Le nombre de zones sur un disque varie suivant le disque (par exemple: 8 pour un HP C2247 et 23 pour un Seagate ST31200W). Ainsi, si on connaît la zone dans laquelle sont placées les données à lire, on pourra connaître le débit utilisé pour lire ces données. De cette façon, le contrôle d'admission est basé sur un débit moyen des lectures et non pas sur un débit minimum. Toutefois, ceci nécessite de pouvoir placer les données explicitement sur le disque. La détection des pistes et le placement des données sont fait par l'algorithme FIXB [GKS96]. L'ordonnancement des lectures est ensuite fait suivant l'algorithme GSS (Grouped Sweeping Scheduling [YCK93], voir section 4.2).

Globalement, Mitra est divisé en 3 composants.

L'ordonnanceur. Il gère l'ordonnancement des requêtes et le contrôle d'admission. Avant de servir une nouvelle requête, il consulte un catalogue pour avoir des informations sur le type de média concerné (bande passante, taille de bloc, localisation des données, etc.). Ensuite, suivant la localisation des données, il utilise soit le système de fichier Everest [GIZ96] pour rapatrier les données du disque, soit le gestionnaire de stockage de masse pour charger les données du stockage tertiaire. Le système de fichier Everest a deux fonctionnalités: tout d'abord il est capable de lire différents fichiers avec une taille de bloc variable ce qui permet de minimiser le gaspillage de bande passante disque lors de lecture de flux de débit différents. Ensuite, dans le cadre d'une hiérarchie de stockage et donc d'écriture fréquentes sur le disques, il permet de minimiser la fragmentation de l'espace disque.

Le gestionnaire de stockage de masse. Il est responsable des opérations de lecture ou écriture concernant le stockage tertiaire.

Le gestionnaire de présentation. Ce composant se trouve chez le client et gère l'arrivée des paquets ainsi que l'affichage de la vidéo.

Comme le montre la figure 18, pour une configuration donnée, on a un processus ordonnanceur actif, un processus de gestion du stockage de masse par périphérique de stockage de masse et un processus gestionnaire de présentation par client. Le contrôle d'admission est déterministe, le serveur assure un débit sans interruption au client (pas d'altération ou de perte d'images).

La tolérance aux pannes est gérée par un système de duplication. Chaque disque est dupliqué dans son intégralité, le disque dupliqué est appelé disque secondaire. Quand une requête de lecture est faite au système de fichier Everest, les données sont lues par les deux disques, le disque principal et le disque secondaire. Si

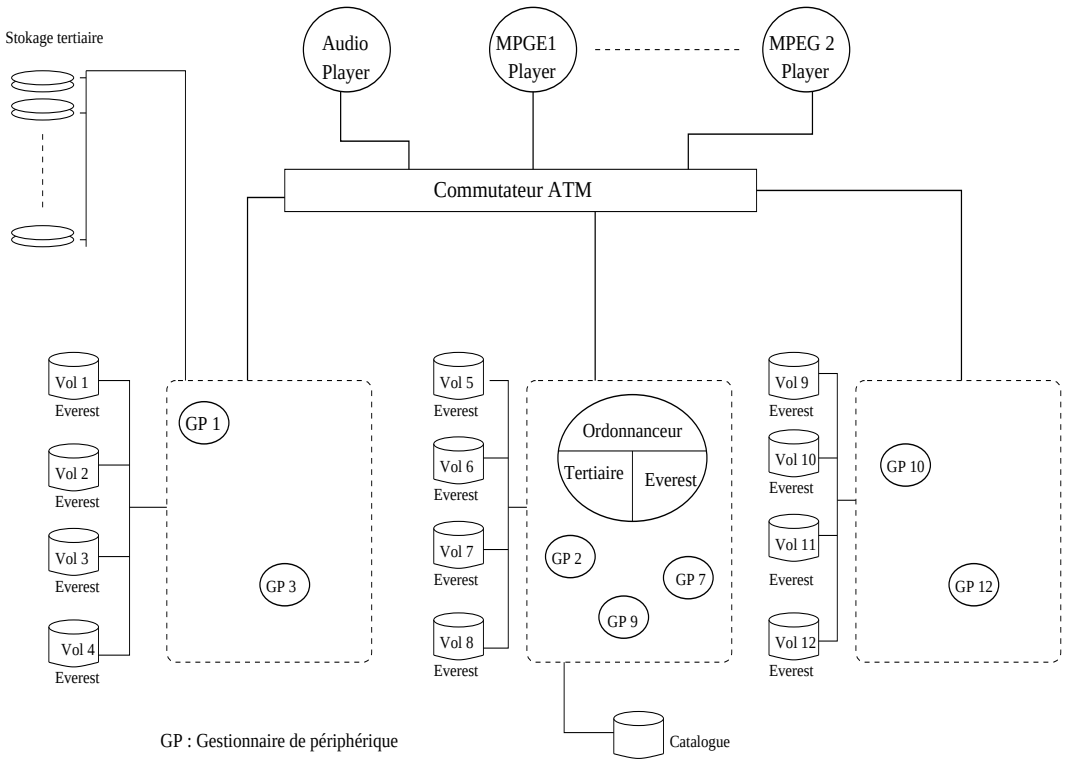

FIG. 18 – *Architecture globale de Mitra.*

le disque principal tombe en panne, les données sont lues dans la mémoire associée au disque secondaire. Il n'y a donc pas de délai occasionné par la panne d'un disque.

Les accès VCR tel que avance et recul rapide sont supportés par Mitra. Pour se faire, des flux rapides sont générés au moment du stockage de la vidéo sur le serveur. Ensuite, lorsqu'un client fait une avance rapide, on ne lit plus les données du flux courant mais son flux rapide associé. Enfin, le support de formats VBR est fait par le découpage d'une vidéo VBR en petites séquences CBR. D'un point de vue performances, Mitra obtient un très bon rapport prix/performance. En revanche, Mitra possède un point unique de panne, l'ordonnanceur sans lequel le serveur ne peut plus fonctionner.

8.3.2 RIO (UCLA): allocation aléatoire

RIO (Randomized I/O, [MSB97]) est le serveur de stockage multimédia développé à UCLA (University of California, Los Angeles) dans le cadre du projet VWDS (Virtual World Data Server). Ce projet a pour but de construire un système multimédia pouvant supporter simultanément différents types d'application multimédia telles que la VOD, visualisation scientifique, navigation virtuelle en 3 dimensions, ou d'autres applications non temps réel.

L'approche de RIO est assez différente des serveurs dédiés uniquement à la vidéo tels que Mitra (cf section 8.3.1). En effet, dans le genre de serveur de ce dernier, les accès aux données sont facilement prédictibles du fait de leur nature continue. Ainsi, des techniques de cache, de pré-chargement, de placement des données, etc. sont utilisées au moment du stockage de la vidéo afin d'en optimiser les accès. RIO ne suppose aucune prédiction des accès aux données. De ce fait, il peut donc supporter tout type de média. Pour ceci, il utilise une technique de placement des données appelée *allocation aléatoire*. Comme son nom l'indique, elle consiste à découper les fichiers en blocs puis à les placer sur les disques de manière aléatoire. Pour maintenir un certain équilibrage de charge et avoir de meilleures performances, une duplication partielle est associée à l'allocation aléatoire permettant d'obtenir des performances comparables aux serveurs vidéo conventionnels tout en supportant une variété de données multimédia. Il existe deux implémentations de RIO, l'une sur une

machine multiprocesseurs Sun Enterprise à 10 processeurs [San98] et une implémentation distribuée sur une grappe de PC [FSM98].

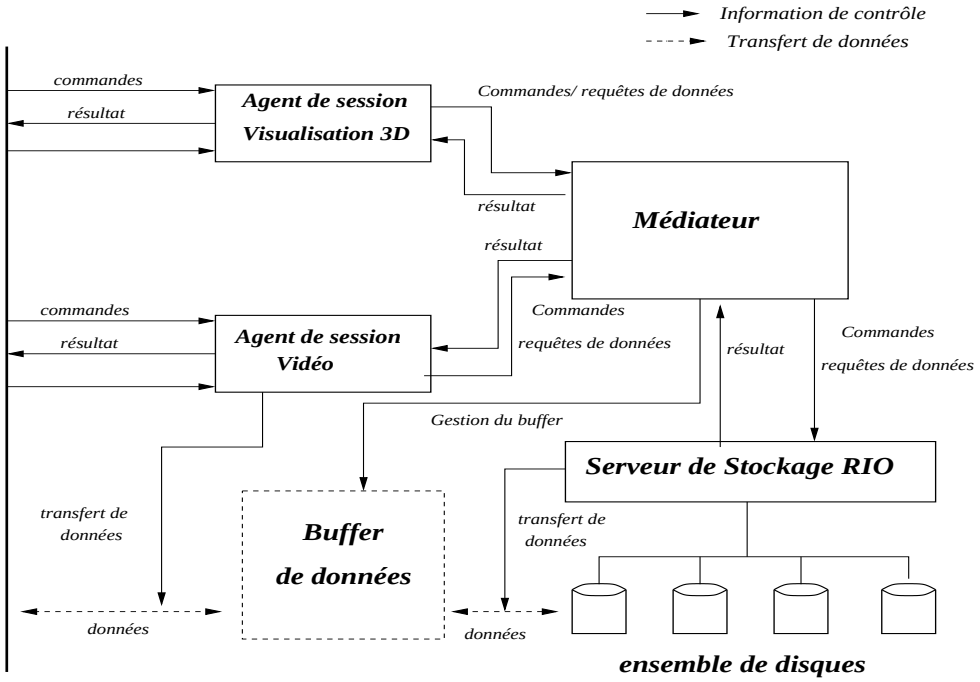

FIG. 19 – *Architecture globale du serveur VWDS.*

Le schéma 19 présente une vue globale du système VWDS en détaillant l'architecture logicielle du serveur. On a 3 composants principaux: le serveur de stockage RIO, le médiateur et les agents de session. Un agent de session est spécifique à une application. Par exemple, l'agent de session pour la visualisation 3D interactive est responsable de l'organisation logique des modèles 3D. Il gère un index spatial pour accéder aux objets et prédit les scènes dont vont avoir besoin les clients dans un futur proche. Cette prédiction est basée sur des informations récoltées sur les utilisateurs précédents, un modèle de déplacement de l'utilisateur et une description de l'angle de vue. Une liste de segments de données correspondant aux scènes sélectionnées est alors soumise au médiateur. Pour l'agent de session vidéo, le principe est beaucoup plus simple. Il génère périodiquement des requêtes aux blocs de données séquentiels d'un objet vidéo en fonction de sa position dans le flux et du débit requis.

Le médiateur quant à lui gère les ressources du système (bande passante disque et réseau, espace mémoire). Il a aussi la charge de la politique d'admission. Le serveur de stockage RIO est ensuite responsable de la lecture sur les disques (distribués sur plusieurs noeuds ou non).

Dans RIO, chaque objet multimédia est composé d'une séquence de blocs de données de taille constante. Chaque bloc est stocké sur un bloc physique aléatoire d'un disque choisi aussi aléatoirement. Cette répartition permet un équilibrage de charge global, mais cependant, il peut parfois se produire une surcharge temporaire sur un disque. Pour adresser ce problème, Muntz *et al.* proposent une réplication partielle des blocs. Ainsi on associe à chaque disque une file des requêtes traitées ou en attente d'être traitées par ce disque. Les nouvelles requêtes sont alors orientées vers le disque ayant la plus petite file d'attente. On peut résumer les avantages de l'allocation aléatoire implémentée dans le serveur RIO par les points suivants.

Support d'application multimédia interactives et hétérogènes. Grâce à l'allocation aléatoire, le serveur peut simultanément des applications interactives dont le comportement n'est pas prédictible, et des applications hétérogènes avec le même schéma de fonctionnement.

Support d'architectures hétérogènes. Le choix d'une allocation aléatoire permet au serveur de s'adapter à une architecture hétérogène. En particulier, ceci permet de gérer un ensemble de disques hétéro-

gènes [SM98]. L'allocation se fait proportionnellement à la capacité de chaque disque, ceci permet de tirer profit des caractéristiques de chaque disque et de ne pas se baser sur le disque limitant.

Extensibilité. Le rajout de disques dans un serveur de stockage nécessite une réorganisation des données pour les distribuer à nouveau sur les disques. Dans le cas d'un stripping classique, ce rajout entraîne un mouvement global de presque toutes les données. Dans le cas de l'allocation aléatoire, cette réorganisation est beaucoup plus simple et consiste à choisir aléatoirement quelques blocs sur les disques déjà présents dans le serveur pour les placer sur le nouveau disque. L'extensibilité du serveur est alors très simple à gérer et très rapide à effectuer.

Fonctionnement en cycles asynchrones. Comme l'ordonnancement disque n'est pas basé sur la structure d'accès de l'application, les disques ne sont pas accédés par cycles synchronisés comme c'est souvent le cas dans les techniques classiques de stripping. L'asynchronisme des cycles permet de ne pas se baser sur le temps de cycle le plus long.

Support du VBR. Dans le fonctionnement de RIO, aucune connaissance a priori n'est connue par le serveur, un flux VBR peut donc être simplement servi comme tout autre flux. Des travaux sont en cours sur la définition adéquate de la politique d'admission dans le cas de flux VBR.

En résumé, l'allocation aléatoire apporte une grande flexibilité par rapport au stripping classique. Les études de performances concernant des flux vidéo [San98] montrent que cette flexibilité n'altére pas ou très peu les performances du serveur par rapport à un stripping classique tirant profit de la séquentialité des accès. Toutefois, dans le cadre du serveur VWDS, on retrouve le même point faible que dans le serveur Mitra: le médiateur géré de manière centralisée reste un point unique de panne.

8.3.3 Symphony (University of Texas, Austin): traitement spécifique pour chaque type de données

Symphony [SGRV98] est un système de fichiers multimédia développé à Austin, Texas. L'objectif principal de ce système est de supporter tout type de données et non pas uniquement des données vidéo. Contrairement à RIO, l'approche choisie ne propose pas un schéma global pour le système de fichiers, mais intègre des techniques de stockage spécifiques à chaque type de données. Dans Symphony, l'approche d'intégration a été adoptée. On a donc un seul serveur englobant des techniques internes spécifiques aux types de donnée concernant des points tel que le placement des données, la tolérance aux pannes, les techniques de cache, etc.

Le schéma global de Symphony est présenté dans la figure 20. Il est constitué de deux couches: une couche dépendante du type de données et une couche indépendante du type de données. La première couche implémente certaines techniques spécifiques tout en se basant sur des outils proposés par la couche inférieure. Ces outils permettent d'accéder aux ressources partagées du serveur et sont au nombre de trois.

Le sous-système disque. Il est responsable de la gestion de l'espace disque et de la bande passante disque. Il est composé de 4 modules.

- **Le gestionnaire de service.** Il ordonnance l'ensemble des requêtes (temps réel ou non) par une combinaison de files d'attente et de divers algorithmes d'ordonnancement (SCAN-EDF, CSCAN). Il gère trois files d'attente chacune dédiée à un type d'accès: les accès périodiques temps réel, les accès apériodiques temps réel et les accès meilleur effort (voir figure 20). Ces trois files alimentent une file globale. La sélection des accès parmi les trois files est dictée par la contrainte temps réel des accès.
- **Le gestionnaire de stockage.** Son rôle est d'allouer ou désallouer des blocs de taille variable sur les disques. Il gère également leur placement.
- **La couche de tolérance aux pannes.** Elle maintient les informations de parité sur les disques et propose des opérations de lecture permettant de reconstruire des données sur un nœud ayant subi une panne ou de lire sans cette reconstruction.

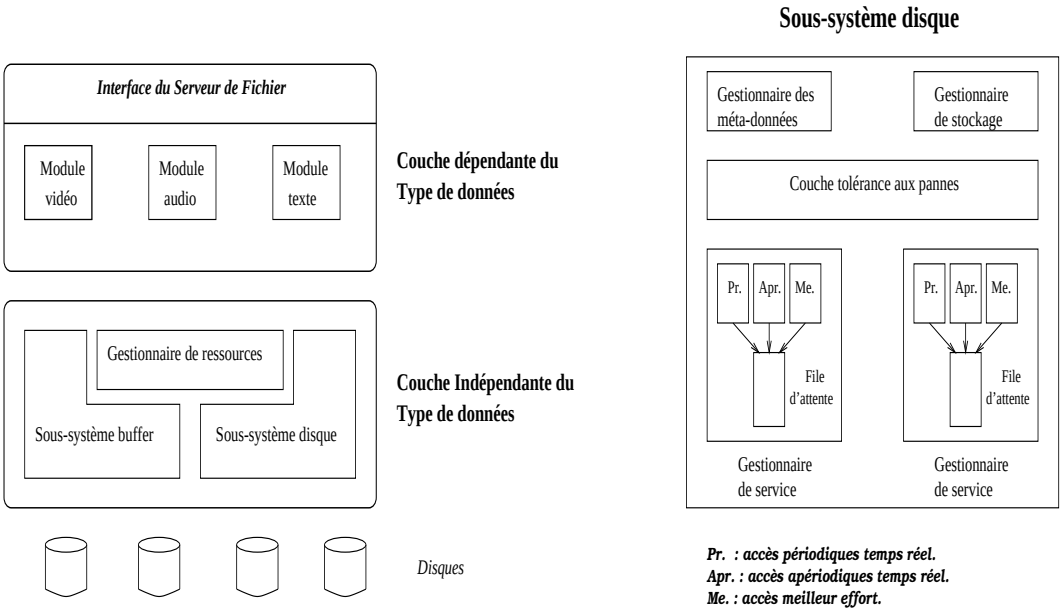

FIG. 20 – *Architecture fonctionnelle en deux couches de Symphony.*

Le gestionnaire des méta-données. Il alloue et désalloue des structures stockant des informations nécessaires à la couche spécifique aux types de donnée.

Le sous-système buffer. Le buffer est partitionné pour que chaque type de données puisse avoir sa propre politique de cache. Ce composant est responsable de la gestion de ces partitions dont la taille peut varier au cours du temps.

Le gestionnaire de ressources. Il permet de réserver des ressources (bande passante disque, espace buffer). Ceci inclut un protocole de négociation de la qualité de service qui permet au client de spécifier ses besoins et une politique d'admission qui détermine si ces besoins peuvent être supportés par l'état actuel du serveur.

Un prototype a été implémenté sur une machine SMP (Ultra Sparc bi-processeur modèle 2700 avec 128 Mo de RAM et 4 disques de 2,1 Go chacun). Les points forts de Symphony résident, d'une part, dans l'approche originale utilisée pour l'ordonnancement des accès disques. Cette approche est basée sur l'exploitation de la nature des données accédées (données statiques ou dynamiques) pour augmenter les performances du système. D'autre part, la coexistence de plusieurs techniques de tolérance et de recouvrement de pannes accroît considérablement la robustesse du serveur face aux pannes des disques.

8.4 Récapitulatif

Comme déjà mentionné, il n'existe pas, dans le domaine des serveurs vidéo, un prototype universel ou meilleur que d'autres, tout comme d'ailleurs il n'existe pas une architecture standard. Ce constat conduit souvent à mettre l'accent sur un développement particulier comme l'ordonnancement des accès disques (Symphony) ou le rapport performance/prix (Mitra) ou encore le placement des données (RIO) etc. Les tableaux 3 et 2 récapitulent les caractéristiques des prototypes de serveurs vidéo présentés précédemment.

9 Perspectives

Durant ces dernières années, le domaine des serveurs vidéo a constitué un domaine de recherche assez actif. Cet engouement est le résultat du concours de deux facteurs importants: d'une part, la disponibilité

	Tiger	Fellini
Architecture	distribuée	mémoire partagée (SMP)
Placement des données	bloc taille variable allocation variable	bloc taille fixe allocation cyclique
Types de donnée supportés	vidéo CBR	données temps réel (CBR) et non temps réel
Tolérance aux pannes	disques et nœuds duplication distribuée	disques parité
Accès interactifs	non	oui
Point fort	pas de point unique de panne	propose deux sortes d'API (temps réel et non temps réel)
Point faible	données CBR	limité à mémoire partagée

TAB. 2 – *Tableau récapitulatif des caractéristiques des prototypes commerciaux présentés.*

	Mitra	RIO	Symphony
Architecture	distribuée	distribuée ou SMP	mémoire partagée (SMP)
Placement des données	bloc de taille fixe Distribution cyclique	bloc taille fixe allocation aléatoire	bloc taille variable allocation variable
Types de donnée supportés	vidéo CBR ou VBR	tout type de données	tout type de données
Tolérance aux pannes	disques et nœuds (réplication des disques)	disques et nœuds réplication des blocs	disques système de parité
Accès interactifs	oui fichier de version rapide	oui	oui
Point fort	bon ratio prix/performance	flexibilité	bonne tolérance aux pannes
Point faible	point unique de panne	point unique de panne	limité à mémoire partagée

TAB. 3 – *Tableau récapitulatif des caractéristiques des prototypes académiques présentés.*

de matériel capable de supporter des applications vidéo, notamment les disques à grande capacité de stockage et à large bande passante ; d'autre part, l'investissement conséquent qu'ont fait les industriels pour le développement de serveurs vidéo.

Une première partie de ce rapport a donc présenté le concept de serveur vidéo en proposant une modélisation d'un serveur vidéo faisant abstraction de tout choix d'architecture matérielle ou d'implémentation (cf. section 2). Cette modélisation permet d'extraire les différents composants fonctionnels du serveur:

le gestionnaire d'interface, responsable de la communication avec le client;

le gestionnaire de ressources, chargé de la gestion interne du serveur;

le gestionnaire de stockage qui effectue les lectures et écritures des données sur l'espace de stockage (qui peut être partagé ou distribué);

le gestionnaire mémoire, responsable du stockage temporaire des données vidéo en provenance du gestionnaire de stockage.

Ensuite, pour chaque composant de cette modélisation, nous avons présenté les techniques de gestion et d'optimisation de leurs fonctions respectives.

La deuxième partie de ce rapport a été consacrée à l'implémentation des serveurs vidéo. En effet, depuis l'apparition des données multimédia dans les années 80, un grand nombre de prototypes ont été développés. Au début de cette partie, nous avons présenté un bref historique de l'apparition et l'évolution de prototypes de serveurs vidéo. Ensuite, dans les section 8.2 et 8.3, nous avons exposé quelques exemples de serveurs vidéo ou multimédia ayant retenus notre attention (Tiger, Fellini, RIO, Mitra et Symphony).

Les objectifs que s'étaient fixés les industriels au début des serveurs vidéo n'ont pas abouti au succès escompté. En effet, la majeure partie des développements concernant les serveurs vidéo s'est concentrée sur les applications VOD. Or, rappelons que ce genre d'applications est constitué de trois parties importantes: le serveur vidéo, le terminal client et le réseau de communication. Si, du point de vue technologique, les deux premières parties ne posent pas de problème, la troisième, quant à elle, ne s'est pas encore affranchie de cette barrière. En effet, le réseau de communication entre le client et le serveur est plutôt un réseau de longue distance (MAN). Or, les réseaux téléphoniques ou à base de faisceaux hertziens n'ont pas la bande passante nécessaire pour transporter de la vidéo. D'autre part, le câblage des clients par des réseaux hauts débits (ATM par exemple) est économiquement infaisable du fait du coût important qu'il représente. Pour ces raisons, les applications VOD n'ont pas eu le succès attendu. Elles devront attendre pour que les coûts des réseaux hauts débits baissent ou bien l'avènement de nouveaux réseaux de communication à plus grande bande passante, ce qui peut arriver très vite avec la généralisation de l'ADSL.

Les développements futurs dans le domaine des applications vidéo vont s'orienter, à notre avis, vers l'ouverture sur Internet. En effet, le succès rapide de cet outil a suscité des investissements non négligeables dans cette direction. Même si, dans l'état actuel, la vidéo ne représente pas un grand pourcentage des accès faits sur Internet, son succès ne fait aucun doute. Selon une étude récente [W+99], sur l'ensemble des accès faits à partir du campus de Washington University, plus de 30% des accès concernent des clips vidéo de tout genre (informations, musique, présentation, etc.), et ce chiffre ne cesse de grandir. Il y a donc là un potentiel important à exploiter et les industriel sont prêts à investir dans ce créneau. En effet, les applications vidéo sur Internet semblent aussi variées que prometteuses (commerce électronique, chaîne de TV, etc.). Cette situation conduira donc, comme on l'a vu précédemment, à s'orienter vers le développement de serveurs vidéo dédiés à ce genre d'application, tout en tenant compte des contraintes qu'impose Internet en termes de limitation de la bande passante du réseau et du nombre important d'utilisateurs.

Les références aux prototypes et aux articles cités peuvent être trouvées à l'adresse suivante :

`http://www.ens-lyon.fr/~abonhomm/video/survey.html`

References

[AHL+91] S. Angebranndt, R.L. Hyde, D.H. Luong, N. Siravara, and C. Schmandt. Integrating audio and telephony in a distributed workstation environement. In *Proceedings of Summer 1991 Usenix Conference*, pages 419–436, June 1991.

[And96] D.B. Andersen. A proposed method for creating vcr functions using MPEG streams. In *IEEE International Conference on Data Engineering (ICDE)*, pages 380–382, February 1996.

[AOG92] A.P. Anderson, Y. Osawa, and R. Govindan. A file system for continuous media. *ACM Transaction on Computer Systems*, 10(4):311–337, November 1992.

[BB96a] C. Bernhardt and E. Biersack. *High-Speed Networking for Multimedia Applications*, chapter The Server Array: A Scalable Video Server Architecture. Kluwer, 1996.

[BB+96b] William J. Bolosky, Joseph S. Barrera, et al. The Tiger video fileserver. In IEEE Computer Society, editor, *Proceedings of the Sixth International Workshop on Network and Operating System Support for Digital Audio and Video*, April 1996.

[BFD97] W.J. Bolosky, R.P. Fitzgerald, and J.R. Douceur. Distributed schedule management in the Tiger video fileserver. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, St Malo, France, October 1997.

[BG88] D. Bitton and J. Gray. Disk shadowing. In *International Conference on Very Large DataBases (VLDB)*, pages 331–338, September 1988.

[Bou92] J-Y. Le Boudec. The asynchronous transfer mode : A tutorial. *Computer Networks and ISDN Systems*, 24:279–309, 1992.

[BPC94] M.M. Buddhikot, G.M. Parulkar, and J.R. Cox Jr. Design of a large scale multimedia storage server. In *Proceedings of the Conference of the Internet Society and the Joint Euopéean Networking Conference (INET'94/JENC5)*, 1994.

[BR96] D. Brubeck and L. Rowe. Hierarchical storage management in a distributed VOD system. *IEEE Multimedia*, 3(3):37–47, 1996.

[C+93] Pei Cao et al. The Ticker TAIP parallel RAID architecture. In *Proceedings of the 1993 International Symposium on Computer Architecture*, May 1993.

[CC96] I. Chen and C. Chen. Threshold-based admission control policies for multimedia servers. *The computer journal*, 39(9):757–766, 1996.

[CCH94] A. Campbell, G. Coulson, and D. Hutchison. A quality of service architecture. *Computer Communication Review*, 24(2):6–27, 1994.

[CCIV99] R. Canonico, G. Capuozzo, G. Iannello, and G. Ventre. MuSA: a scalable multimedia server based on clusters of SMPs. In *ACM International Conference on SuperComputing- ICS'99, Proceedings of the 1999 Workshop on Cluster-Based Computing (WCBC'99)*, pages 41–45, Rhodes, Grece, June 1999.

[CGM97] E. Chang and H. Garcia-Molina. Effective memory use in media server. In *International Conference on Very Large DataBases (VLDB)*, pages 496–505. Athens, Greece, August 1997.

[Che90] Ann Chervenak. Performance measurements of the first RAID prototype. Technical report, Department of computer science, University of Berkeley, 1990.

[Che98] Ann Chervenak. Challenges for tertiary storage in multimedia servers. *Parallel Computing*, 24(1):157–176, January 1998.

[CK96] M. Chen and D. Kandlur. Stream conversion to support interactive video playout. *IEEE Multimedia*, 3(2):51–58, 1996.

[CKY95] M. Chen, D.D. Kandlur, and P.S. Yu. Storage and retrieval methods to support fully interactive playout in a disk-array-based video server. *Multimedia Systems*, 1995.

[CL96] H. Chen and T. Little. Storage allocation policies for time-dependent multimedia data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 8(5):855–864, 1996.

[CLY96] M. Chen, C. Li, and P. Yu. A framework for experimenting with QoS for multimedia services. In *IS&T/SPIe Multimedia Computing and Networking*, pages 29–31, January 1996.

[CZ96] E. Chang and A. Zakhor. Cost analyses for VBR video servers. In *IS&T/SPIE International Symposium on Electronic Imaging : Science and Technology*, volume 2667, pages 29–31. San Jose, California, January 1996.

[DS96] A. Dan and D. Sitaram. *Multimedia Information Storage and Management*, chapter Session Scheduling and Resources Sharing in Multimedia Systems. Kluwer, 1996.

[DS97] A. Dan and D. Sitaram. Multimedia caching strategies for heterogeneous application and server environment. *Multimedia Tools and Applications*, 4(3):279–312, 1997.

[DSS96] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video servers. *Multimedia Systems*, 4(3):112–121, 1996.

[DVV94] D. Deloddere, W. Verbiest, and H. Verhille. Interactive video-on-demand. *IEEE Communications Magazine*, 32(5):82–88, May 1994.

[FBW96] C.S. Freedman, J. Burger, and David J. De Witt. SPIFFI: A scalable parallel file system for the intel paragon. *IEEE Transaction on Parallel and Distribted Systems*, 7(11):1185–1200, November 1996.

[FSM98] F. Fabbrocino, J.R. Santos, and R. Muntz. An implicitly scalable real-time multimedia storage server. In *Second Workshop on Distributed Interactive Simulation and Real-Time Applications (DIS-RT98)*, pages 92–101, June 1998.

[Gal91] D. Le Gall. MPEG : A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):47–58, April 1991.

[GDS95] S. Ghandeharizadeh, A. Dashti, and C. Shahabi. Pipelining mechanism to minimize the latency time in hierarchical multimedia storage managers. *Computer communication*, 18(3):170–184, 1995.

[GIZ96] S. Ghandeharizadeh, D. Ierardi, and R. Zimmermann. An online algorithm to optimize file layout in a dynamic environment. *Information Processing Letters Journal*, 57:75–81, 1996.

[GIzS98] M.N. Garofalakis, Y.E. Ioannidis, B. Özden, and A. Silberschatz. Throughput-competitive admission control for continuous media databases. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles Of Databases Systems*, pages 79–88. Seattle, Washington, June 1998.

[GKS96] S. Ghandeharizadeh, S.H. Kim, and C. Shahabi. *Multimedia Information Storage and Management*, chapter Placement of Continuous Media in Multi-Zone Disks. Kluwer, 1996.

[GLP98] L. Golubchik, J. Lui, and M. Papadopouli. A survey of approaches to fault tolerant design of VOD servers: Techniques, analysis and comparison. *Parallel Computing*, 24(1):123–155, 1998.

[GM98] S. Ghandeharizadeh and R. Muntz. Design and implementation of scalable continuous media servers. *Parallel Computing*, 24:91–122, 1998.

[GTF+87] S. Gibbs, D. Tsichritzis, A. Fitas, D. Konstantas, and Y. Yeorgaroudakis. Muse: A multi-media filing system. *IEEE Software*, 4(2):4–15, March 1987.

[GVK+95] D.J. Gemmel, H.M. Vin, D.D. Kandlur, P.V. Rangan, and L. Rowe. Multimedia storage servers: A tutorial and survey. *IEEE Computer*, 28(5):40–49, November 1995.

[GZS+97] S. Ghandeharizadeh, R. Zimmermann, W. Shi, R. Rejaie, D. Ierardi, and A.W Li. Mitra : A scalable continuous media server. *Multimedia Tools and Applications Journal*, 5(1):79–108, July 1997.

[Has93] Roger L. Haskin. The Shark continous media file server. In *Proceedings of the IEEE Computer Society International Conference, COMPCON'93*, 1993.

[HLL+95] J. Hsieh, M. Lin, J.C L. Liu, D.H.C. Du, and T.M. Ruwart. Performance of a mass storage system for video-on-demand. In *Proceedings of the 14th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'95)*, pages 771–778, Los Alamitos, CA, USA, June 1995. IEEE Computer Society Press.

[HMS99] J.H. Hartman, I. Murdock, and T. Spalink. The Swarm scalable storage system. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS'99)*, June 1999.

[Hop90] A. Hopper. Pandora - an experimental system for multimedia applications. *ACM Operating Systems review*, 24(2):19–34, April 1990.

[JSC97] D. Jadav, C. Srinilta, and A. Choudhary. Batching and dynamic allocation techniques for increasing the stream capacity of an on-demand media server. *Parallel Computing*, 23(12):1727–1742, 1997.

[JSCB95a] D. Jadav, C. Srinilta, A. Choudhary, and B. Berra. Techniques for scheduling I/O in a high performance multimedia-on-demand server. *Journal of Parallel and Distributed Computing*, 30:190–203, 1995.

[JSCB95b] D. Jadav, C. Srinilta, A. Choudhary, and P. Berra. Design and evaluation of data access strategies in a high performance multimedia-on-demand server. In *Second International IEEE Conference on Multimedia Computing and Systems*, pages 286–291, May 1995.

[KDST95] M. Kienzle, A. Dan, D. Sitaram, and W. Tetzlaff. Using tertiary storage in video-on-demand servers. In *COMPCON'95*, pages 217–224, March 1995.

[KRT95] M. Kamath, K. Ramamritham, and D. Towsley. Continuous media sharing in multimedia database systems. In World Scientific, editor, *DAtabase Systems for Advanced Applications (DASFAA)*, pages 79–86. Singapore, April 1995.

[L+92] E. Lee et al. RAID-II: A scalable storage architecture for high-bandwidth network file service. Technical Report UCB//CSD-92-672, Department of computer science, university of Berkeley, October 1992.

[Lee98] Kack Y.B. Lee. Parallel video servers: a tutorial. *IEEE Multimedia*, pages 20–28, June/April 1998.

[LOP94] A. Laursen, J. Olkin, and M. Porter. Oracle media server: Providing consumer based interative access to multimedia data. In *Proceedings of the 1994 ACM SIGMOD*, pages 449–456, Minneapolis, May 1994.

[LS93] P. Lougher and S. Shepherd. The design of a storage server for continuous media. *The Computer Journal*, 36(1), 1993.

[LV94] T. Little and D. Venkatesh. Prospects for interactive video-on-demand. *IEEE Multimedia*, 1(3):14–24, Fall 1994.

[MKK95] F. Moser, A. Kraisse, and W. Klas. L/MRP: A buffer management strategy for interactive continuous data flows in a multimedia DBMS. In *International Conference on Very Large DataBases*, pages 275–286. Zurich, Switzerland, September 1995.

[MNz+96] C. Martin, P.S. Narayanan, B. Özden, R. Rastogi, and A. Silberschatz. *Multimedia Information Storage and Management*, chapter The Fellini Multimedia Storage Server. Kluwer Academic, 1996.

[Mor90] Y. Mori. Multimedia real-time file system. Matshushita Electric Industrial Co., February 1990.

[MSB97] R. Muntz, J.R Santos, and S. Berson. RIO : A real-time multimedia object server. *ACM Performance Evaluation Review*, 25(2):29–35, September 1997.

[Nat95] Krishnan Natarajan. Video servers take root. *IEEE Spectrum*, pages 66–69, April 1995.

[NLO95] M.N. Nelson, M. Linton, and S. Owicki. A highly available, scalable ITV system. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, pages 54–67, December 1995.

[NY94] R. Ng and J. Yang. Maximizing buffer and disk utilizations for news-on-demand. In *International Conference on Very Large DataBases*, pages 451–462. Santiago de Chile, September 1994.

[Pan97] H. Pang. Tertiary storage in multimedia systems: Staging of direct access. *Multimedia Systems*, 5(6):386–399, 1997.

[PGS88] D. Patterson, G. Gibson, and M. Satyanarayanan. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD)*, pages 81–94, Chicago, IL, June 1988.

[Red96] N. Reddy. *Multimedia Information Storage and Management*, chapter Improving the Interactive Responsiveness in a Video Server. Kluwer, 1996.

[RV91] V. Rangan and H. Vin. Designing file systems for digital video and audio. In *Proceedings of the 13th Symposium on Operating Systems*, pages 81–94, October 1991.

[RV93] P.V. Rangan and H.M. Vin. Efficient storage techniques for digital continuous multimedia. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):564–573, August 1993.

[RW94] A. Reddy and J. Wyllie. I/O issues in a multimedia system. *Computer*, 27(3):69–74, 1994.

[RZ95] D. Rotem and L. Zhao. Buffer management for video database systems. In *International Conference on Data Engineering (ICDE)*, pages 439–448. Taipei, Taiwan, March 1995.

[San98] Jose Renato Goncalves Santos. *RIO: A Universal Multimedia Storage System Based on Random Allocation and Block Replication.* PhD thesis, University of California, Los Angeles, 1998.

[SG97] W. Shi and S. Ghandeharizadeh. Buffer sharing in video-on-demand servers. *SIGMETRICS Performance Evaluation Review*, 25(2):13–20, 1997.

[SGRV98] P.J. Shenoy, P. Goyal, S. Rao, and H.M. Vin. Symphony: An integrated multimedia file system. In *Proceedings of ACM/SPIE Multimedia Computing and Networking 1998 (MMCN'98)*, pages 124–138, San Jose, January 1998.

[SGV95] P. Shenoy, P. Goyal, and H. Vin. Issues in multimedia server design. *ACM Computing Survey*, 27(4):636–339, December 1995.

[SKKP99] Y.S. Son, O.Y. Kwon, T.G. Kim, and A.A. Park. A high performance VOD server and its I/O scheme on PC-clustering environment. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, Las Vegas, June 1999.

[SLM97] Olav Sandsta, Stein Langorgen, and Roger Midtstraum. Video server on an ATM connected cluster of workstations. In *Proceedings of the XVII International Conference of the Chilean Computer Society (SCCC'97)*, pages 207–217, Valparaison, Chile, November 1997.

[SM98] J.R. Santos and R.R. Muntz. Performance analysis of the RIO multimedia storage system with heterogeneous disk configurations. In *6th ACM International Multimedia Conference (ACM Multimedia 98)*, pages 303–308, Bristol, United Kingdom, September 1998.

[Sun89] Sun Microsystems. *Multimedia File System*, August 1989. Software Release.

[SV99] P. Shenoy and H. Vin. Efficient support for interactive operations in multi-resolution video servers. *Multimedia Systems*, 7(3):241–253, May 1999.

[SY98] S. Shachnai and P. Yu. On analytic modeling of multimedia batching shemes. *Performance Evaluation*, 33(3):201–213, 1998.

[TFC+85] R.H. Thomas, H.C. Fosdick, T.R. Crowley, R.W. Schaaf, R.S. Tomlinsin, V.M. Travers, and G.G. Robertson. Diamond: A multimedia message system built on a distributed architecture. *IEEE Computer*, 18(12):65–78, December 1985.

[TMDV96] R. Tewari, R. Mukherjee, D.M. Dias, and H.M. Vin. Design and performance tradeoffs in clustered video servers. In *the IEEE international Conference on Multimedia Computing and Systems (ICMCS'96)*, pages 144–150, May 1996.

[TP97] P. Triantafillou and T. Papadakis. On-demand data elevation in hierarchical multimedia storage servers. In *International Conference on Very Large DataBases (VLDB)*, pages 226–235, 1997.

[VGGG94] H.M. Vin, A. Goyal, A. Goyal, and P. Goyal. An observation based admission control algorithm for multimedia servers. In *IEEE International Conference on Multimedia Computing and Systems (ICMCS'94)*, pages 234–243, May 1994.

[VPGG94] H.M. Vin, P.Goyal, A. Goyal, and A. Goyal. A statistical admission control algorithm for multimedia servers. In *ACM Multimedia'94*, pages 33–40, October 1994.

[VV99] Pascal Vandeputte and Jean-Marc Vescovo. SCI based scalable video on demand server. In *Conference Proceedings of SCI Europe'99*, pages 95–97, September 1999. Siemens ICP Computers.

[W+99] A. Wolman et al. Organization-based analysis of web-object sharing and caching. In *in 2nd USENIX Conference on Internet Technologies and Systems*, October 1999.

[WA85] J. Wong and M. Ammar. Analysis of broadcast delivery in a videotex system. *IEEE Transactions on Computers*, 34(9):863–866, 1985.

[WL97] P.C. Wong and Y.B. Lee. Redundant array of inexpensive servers (RAIS) for on-demand multimedia services. In IEEE Computer Society Press, editor, *Proceedings of ICC'97*, pages 787–792, Los Alamitos, CA, 1997.

[YCK93] P.S. Yu, M.S. Chen, and D.D. Kandlur. Grouped sweeping scheduling for DASD-based multimedia storage management. *Multimedia Systems*, 1(1):99–109, January 1993.

[YWS96] P. Yu, L. Wolf, and H. Shachnai. *Multimedia Information Storage and Management*, chapter Scheduling Issues in Video-On-Demand Systems. Kluwer, 1996.

[Zim98] Roger Zimmermann. *Continuous media placement and scheduling in heterogeneous disk storage systems.* PhD thesis, University of Southern California, December 1998.

[zRS95] B. Özden, R. Rastogi, and A. Silberschatz. Research issues in multimedia storage servers. *ACM Computing Surveys*, 27(4):617–620, 1995.

[zRS96] B. Özden, R. Rastogi, and A. Silberschatz. Buffer replacement algorithms for multimedia databases. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, June 1996.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399